

TI3800 BACHELOR PROJECT

BUILDING A GUI FOR NUNA'S MISSION CONTROL

FINAL REPORT



AUTHORS

Isha van Baar
Zakaria el Bakkali
Jurian van Dalfsen
Eveline van der Schrier
Lodewijck Vogelzang

I.J.vanBaar@student.tudelft.nl
Z.elBakkali@student.tudelft.nl
J.vanDalfsen@student.tudelft.nl
E.D.vanderSchrier@student.tudelft.nl
L.G.J.Vogelzang@student.tudelft.nl

COACH

Nicolas Dintzner

N.J.R.Dintzner@tudelft.nl

CLIENT

Rune van der Meijden

rune@nuonsolarteam.com

FORMER APPLICATION DESIGNER

Noud Brasjen

noud.nuna6@gmail.com

COURSE COORDINATORS

Martha A. Larson
Felienne Hermans

M.A.Larson@tudelft.nl
F.F.J.Hermans@tudelft.nl

FOREWORD

This report serves as the official end deliverable for the TI3800 Bachelor Project. This project, also referred to as “Bachelor Eindproject” or “BEP”, is the course that finishes the Bachelor of Computer Science at Delft University of Technology. It is carried out by groups that are given the assignment to work together like a software development team, producing a real product to be used by a real client, solving a real-world problem. In this sense all knowledge that is gained in the three year studies is needed to successfully finish this project.

In the case of our team, we have done a very practical project at the Nuon Solar Team. In short: our goal was to reengineer an existing telemetry application for the chase car of the Nuon Solar Car.

The contents of the Bachelor Project were revealed on February the 17th. After that, the project would officially start on the 21th of April (Week 17). However, as this team needed more time for both research and development, it was decided to start on the 11th of March (Week 11). Furthermore, the project has unofficially been extended with one week, to be finished at the 11th of July (Week 28).

During these weeks our team has been present full time at the Delft University of Technology, mostly at the Faculty of Electrical Engineering, Mathematics and Computer Science and often at the Nuon Solar Team workplace.

We would like to thank the following persons in particular:

- Nicolas Dintzner, for being the best coach we could imagine: critical, supportive, smart and with a great sense of humor. We could not have wished for anyone better.
- Rune van der Meijden, for giving us helpful feedback, support and stories about the Nuon Solar Team.
- Noud Brasjen, for being present at numerous extensive sessions to give us a deeper understanding of his code and our purpose.
- Bart Koek, for doing the best he could to arrange us work spots and valuable time with the Nuna solar car.
- Arie van Deursen, for giving us a lot of support on making our code extensible, maintainable and simply looking more structured than ever before.
- Martha Larsson, for always answering our urgent questions about applying the general Bachelor Project rules to our special project.

Isha van Baar,
Zakaria el Bakkali,
Jurian van Dalfsen,
Eveline van der Schrier,
Lodewijck Vogelzang

Delft, Zuid-Holland, the Netherlands, July 2014

TABLE OF CONTENTS

Foreword	2
Table of contents.....	3
Summary	5
1. Introduction.....	6
2. Methodology	7
2.1 Strategy.....	7
2.2 Planning.....	7
2.3 Project roles.....	8
2.4 Meetings	8
2.5 Resources	9
2.6 Tools	9
2.6.1 Programmer applications	9
2.6.2 Designer applications	10
2.6.3 Project management applications	11
3. Research Phase	12
3.1 Problem definition.....	12
3.2 Problem analysis.....	12
3.2.1 User requirements	12
3.2.2 System requirements	13
3.2.3 Features	13
3.3 System design and structure.....	16
3.4 Language and framework research.....	22
3.4.1 Programming language	22
3.4.2 GUI frameworks	23
3.4.3 Connection frameworks.....	24
4. Development phase.....	25
4.1 Implementations	25
4.2 Extensibility of implementations.....	27
4.3 Extra functionalities	29
4.4 Testing in development phase	30
4.5 Development challenges.....	30
5. Concluding phase	32
5.1 Testing in concluding phase	32
5.2 SIG code evaluation	32
5.2.1 First evaluation.....	32
5.2.2 Second evaluation	33
5.3 Documentation of final product.....	33
6. Results.....	35
6.1 Package contents	35
6.2 Layout of the application.....	35
6.2.1 Overview	35
6.2.2 Sensor view	36

6.2.3	Strategy view	41
6.2.4	Top pane	43
6.2.5	Bottom pane.....	45
7.	Conclusion.....	46
7.1	Problem requirements	46
7.1.1	Testing days.....	46
7.1.2	Weekly meetings.....	46
7.1.3	Requirements.....	46
7.2	Product Validation.....	47
7.3	Learnings.....	47
7.3.1	Learnings from the research phase	47
7.3.2	Learnings from the development phase	48
7.3.3	Learnings from the concluding phase	48
7.4	Used knowledge.....	49
8.	Discussion and recommendations.....	50
8.1	Product discussion	50
8.2	Process reflection	50
8.3	Recommendations	50
9.	References	
	Appendix 1: Plan of Action.....	
	Appendix 2: Research Phase Report	
	Appendix 3: Project Logs	
	Appendix 4: Testing Plan	

SUMMARY

In the history of the World Solar Challenge, a six day challenge from Darwin to Adelaide, one car has proved to be dominating: the Nuon Solar Car. A telemetry application played an important role in this, but needed to be updated. This is realised by the new Mission Control application project.

The project started with a research phase, a project strategy was determined and a planning was made. Multiple tools, programming languages and packages were investigated to solve the client's problem: creating a new, extendable application.

After all the requirements were listed for the application, the team started with designing the structure of the application. Trying to design the structure of the application similar to the previous application, but in a way that the application is easily extendable.

The next phase, the development phase was the most important phase of the project. In this phase the product is designed, created and tested. Some extra functionalities, which the Nuon solar team did not ask for have been implemented as well. The client is happy about these extra functionalities and they certainly help in the overall experience of the interaction with the application.

In comparison with the development phase, testing in the concluding phase consisted more of system testing. This was to make sure the application worked as desired. There have been a few testing days where there was the possibility of testing the application as if in a real race. Not only was this a great experience, but also very important for fixing the last bugs and improving the functionality of the application.

Since the formal client was someone who would in reality race no more in the Mission Control chase car, there were different stakeholders in charge to get connected to the product. Some successors should be able to fully understand our application, while others should only be able to use it. For each target group, a manual has been written with exactly the contents that should concern them. All of these manuals will be included in the final documentation package that will be delivered next to the application code package. The manuals, sorted on their respectively mentioned target group include:

- *User Manual*
- *Extensibility Manual*
- *Technical Manual*

1. INTRODUCTION

In the history of the World Solar Challenge, a 6 day challenge from Darwin to Adelaide, one car has proved to be dominating: the Nuna Solar Car. This car is redesigned every two years by students of the Delft University of Technology who call themselves the Nuon Solar Team. In the race that counts 3000 kilometres, the Nuon Solar Team has won five times.

At every update, new stunning specifics are revealed. The car can lose weight, have a better engine or simply have a better mechanism to gain or divide solar energy from its large solar cell surface. However, often one factor for success is underestimated: forming the race strategy.

To successfully finish the race it is essential to be able to make quick strategic decisions. The team has a lot of freedom in deciding on Nuna's speed and (charge) stops. While in a normal race, these would be simple decisions, for a solar car like Nuna it turns complicated very fast. Not only do you have to keep track on your competitors but monitoring data of what happens inside the Nuna is crucial for well thought decisions.

What if all that data would be monitored in a telemetry application used in a chase car? More importantly: what if it turns out that this application is not extendible or maintainable at all and that all knowledge from its code is owned by only one person, responsible for the Strategy?

This is exactly the problem that the Nuon Solar Team has faced and that our team has solved in this Bachelor Project. Luckily, the solar team will have one more opportunity to test new equipment like the application for the chase car. This September, there will be another solar challenge: the South Africa Solar Challenge.

The application is named *Mission Control* after the name for the chase car. It will be used during the South Africa Solar Challenge. It has been developed in three phases: the research phase, the development phase and the concluding phase. This report has been divided in respect these three phases, to describe what we have done in each phase.

Also, it is important to mention that we have made other documentation that specifically targets our users and successors. This was part of our deliverables. We included these three different manuals as separate appendices next to this report. There are the *User Manual*, the *Extensibility Manual* and the *Technical Manual*.

The overview of this report is as follows. First our project methods are described (section 0) and compared to what we have proposed in the beginning of the research phase in our *Plan of Action* document (). This consists of our strategy (section 2.1), planning (section 2.2), project roles (section 2.3), meetings (section 2.4), resources (section 2.5), tools (section 2.6). Then our research phase is described (section 0) by defining and analysing the problem (section 3.1, section 3.2) and describing our system design (section 3.3) and language and framework choice (section 3.4). These could also be found in more detail in the *Research Phase Report* ().

Thirdly, our development phase is described (section 4). In the implementations (section 4.1) our developed code will be given in terms of models and its most important packages. Also, as an important part of our work consisted of making the code extensible and maintainable, our measures to ensure this are given (section 4.2). After that our implemented extra functionalities (section 4.3) and testing methods in the development phase (section 4.4) are described. The development phase is finished by telling about its challenges (section 4.5).

In the concluding phase (section 0) we have also found ourselves doing a lot of tests, but these tests were sometimes of a different consistency (e.g. doing more validation tests). This is why the tests in the concluding phase are also described in a separate part (section 5.1). In the final moments of our project our team also had to react to the SIG code evaluation (section 5.2) and to document the three different manuals for our successors (section 5.3).

Fourthly, we have included our results (section 6) by discussing our package contents (section 6.1) and the layout of the application (section 6.2). In the conclusion (section 0), our team critically looks at our problem requirements (section 7.1) and shows that we have met them (section 7.2). Also, our learnings from this project are given (section 7.3). In the section hereafter (section 0), we discuss what we could have done better in terms of product (section 8.1) and process (section 8.2). Also, some recommendations for new teams are formed (section 8.3).

Finally, our references are given (section 9) and several appendices are included.

2. METHODOLOGY

This section describes the method of working during the entire project. The team strategy is explained and the projects planning and all used tools are given. In each section a comparison has been made between the initial methods of the *Plan of Action* () and the actual realization of the project.

2.1 STRATEGY

Before the start of the research phase our team decided to use *Agile Management* as our main methodology of working (*Plan of Action*). One of the derived management types of Agile Management is *SCRUM*, which our team decided to use. The team planned to work with SCRUM iterations of one week where each sprint started at Monday morning with a Weekly SCRUM meeting in which all items for that week were decided. Each sprint resulted in a prototype on Friday with the newly implemented functionality.

Besides that it was also decided to make a global planning of the development phase (section 2.2). This planning was made to ensure that all critical requirements (section 3.2) were implemented at the end of the development phase.

2.2 PLANNING

At the start of this project, the team divided the project in three phases: the research phase, the development and the concluding phase (firstly called test phase). It was also decided that a high level planning during the research phase was necessary to compare our progress against a general planning, the *planning of tasks* (Table 1).

	Research phase			Exams	Development phase										Test phase		Extension
Product Backlog / Sprint week	11	12	13	14-16	17	18	19	20	21	22	23	24	25	26	27	28-29	
Documenting List of Requirements																	
Documenting Plan of Action Report																	
Reverse engineering of current application																	
Researching Java GUI frameworks																	
Researching Java real-time connections																	
Documenting Research Phase report																	
Modelling the new application																	
Creating a framework																	
Connecting application to mobiboxx																	
Sending, requesting mobiboxx data																	
Showing data in some way in framework																	
Creating overview panel																	
Creating battery panel																	
Creating power flow panel																	
Testing application with driving solar car																	
Creating strategy panel																	
Creating remaining panels																	
Sending code to SIG for evaluation																	
Creating remaining features																	
Implementing SIG code feedback																	
Composing Bachelor Project Report																	
Testing thoroughly																	
Booking presentation room and inform																	
Resending code to SIG																	
Presenting the final product																	

Table 1: Initial planning of tasks

The team kept track of the progress by filling in a similar, empty version of this table, the *journal of tasks*. Also, we managed to keep track of our work hours. All these logs of the project could be found in the appendices ().

The differences between the planning created during the research phase and the *journal of tasks* are mainly in the development phase:

- The team underestimated the time to complete the tasks that were planned for week 17. Creating the framework of the application and connecting to the Nuna's Mobiboxx (section 3.1) turned out to take two weeks
- There was a different order in which the panels were implemented. Since the Overview panel (section 6.2.2) contained mainly information from different other tabs, the team decided to implement the other tabs first so that their information

- The strategy tab also took more time than expected
- There were some implemented functionalities that were unlisted on the planning of tasks document that were formed during the project. These could be found in section 4.3

2.3 PROJECT ROLES

At the beginning of the development phase there was another unforeseen change to our initial methodology. It was found to be useful that each member, apart from his or her usual tasks, became responsible over one important aspect of the project. These different tasks (Table 2) were found very useful.

Isha van Baar	Project leader
Zakaria el Bakkali	Responsible for neatness of the code
Jurian van Dalfsen	Responsible for the functionality of the code and creating prototypes
Eveline van der Schrier	Responsible for the testing of the code
Lodewijck Vogelzang	Secretary

Table 2: Member roles during the development phase

The responsibilities of the project leader were to prepare team meetings (2.3), scheduling client and coach appointments and guiding the team in the process of the project.

The second role includes the code neatness. This task was defined by ourselves with the following guidelines:

- Unit complexity
Absence of long lines of code or long blocks of code
- Module decoupling
Absence of parts of the code that are often called
- Clear documentation
Correct, complete and clear Javadoc comments in the code
- Clear names
Clear class, method and object names

The role that is in charge of the functionality of the code firstly measured the presence and correct functioning of the backlog items that are planned for the sprint. When all items were correctly implemented and tested at the end of the sprint, a prototype was created. Due to the project not being correctly configured as a Maven project, the first five prototypes were generated by *Eclipse* by exporting the project as an *executable jar file* instead of adding all dependencies in a Maven generated *executable jar file*.

The responsibility of the testing of the code included the checking the unit tests of the code for presence, correctness and clearness. When the present tests run correctly and were clearly commented, the test coverage was calculated to make sure the tested class was sufficiently tested. Since there were some difficulties with testing (section 4.4), this quickly became a challenging role.

It was agreed that these responsibilities should not include finishing another team member's work. The people fulfilling these rules are responsible for pointing out when some part of the code is not yet finished.

Lastly, the secretary was responsible for taking notes during meetings. The secretary was also the contact point for people outside the team. The secretary kept the planning up-to-date and was therefore responsible for the planning of tasks and work hours document.

2.4 MEETINGS

Originally several internal meetings were planned per sprint (*Plan of Action*). First of all, on Monday morning the sprint would be started with a Weekly SCRUM where the team would select the backlog items that should be finished during the sprint. It was also planned that during the sprint the team would have two daily meetings at the start and the end of the day to keep track of the progress. In these meetings, the members could present their progress and problems can be quickly spread and discussed.

Besides the internal meetings, the team planned multiple external meetings. Each week there would be a meeting with both the coach (to ask questions, receive feedback on work and problems) and the client (to present prototypes, receive feedback on work).

Although the initial planning of the meetings was very tightly followed, there were some small differences:

- Since the same work space was used during the sprint days, the progress and problems were often instantly discussed. Therefore the meeting at the end of the day was sometimes unnecessary since no new progress or problems had to be discussed.
- The team also met several times with Arie van Deursen to receive help with configuring the project as a Maven project (section 2.6.1)

2.5 RESOURCES

In the *Plan of Action* document, some resources were mentioned to be of critical importance to the team. Examples were access to the most recent version of the Nuna Solar Car, Nuna 7. Also, access to the sensor devices was said to be very important.

Unfortunately, these resources were not always found to be present. At some point, this became problematic since we had not thought of capturing dummy data before as we had not finished reading all data yet. The point that we realized that Nuna 7 (and to some extent also Nuna 6) would very often be absent, even at the time of our concluding phase, was simply too late. This has also been the reason why the team has not experienced a test drive with the Nuna 7.

From that point on, we tried to use Nuna 6 as much as possible as a replacement to Nuna 7. However, as Nuna 6 was also very often absent. Also, in Nuna 6, some parts were not connected or present, which made developing and testing even harder.

However, we do have tried to work around these obstacles as much as we could and simulate the execution environment in our testing environment. More about these issues could be found in the *Development challenges* (section 4.5)

2.6 TOOLS

The team used certain tools to aid the process of the project. These could be divided into programmer applications (section 2.6.1), designer applications (section 2.6.2) and project management tools (section 2.6.3). All tools are sorted in the list based on how much we have used them.

2.6.1 PROGRAMMER APPLICATIONS

Computing environments:

- Eclipse
The team decided to use *Eclipse* [1] as software IDE for creating the product. All team members were familiar with this tool and have used it before. *Eclipse* offers functionality for development in Java. Most of the applications development was done in *Eclipse Kepler*.
- Visual Basic 6
The previous Mission Control application was implemented in *Visual Basic 6*. To run the program for references and to open the source code to for instance copy formulas, the team had to install the *Visual Basic 6* tool. This was more difficult than expected since Microsoft has stopped supporting it [2]. Besides opening and running the previous application, this application has not been used.
- Matlab
The application uses longitude and latitude coordinates to specify the route of the race. The developed application uses *H2* databases with the route coordinates as input for the race. However the previous application used *Matlab* files as input. Since there was not yet any route data available in the required format, the team used *Matlab* [3] to create these input databases themselves.

Plugins/code tools:

- Maven
Maven [4] is a tool for continuous integration. It was designed to give all Java projects generally the same structure. With that in mind, building and documenting projects should become a lot easier. Each project has a *project object model (POM)* file in which the dependencies of the project are noted. The dependencies do not have to be added to the project manually, but can all be installed by this tool. *Maven* also offers functionality to create an *executable jar* file of the project, where certain resources of the project can be placed outside of this *jar* file. The team used this tool before and was satisfied with its functionality. However, designing a *Maven* structure for this application took us longer than expected since our three projects had to be configured into one general one, with the generally defined project structure of *Maven* applied to it. Arie van Deursen helped us a great deal with this issue.

- JUnit
JUnit [5] has been a very important tool for the project as it was used for unit testing. It allowed for a very easy way of creating test methods for functions and running the automated unit tests. The tool is used together with the *Mockito* package, which allows for the mocking of objects, which greatly increases the testability of objects.
- EclEmma
EclEmma [6] is a tool that is used for calculating the test coverage of a project. Used together with *JUnit*, these tools made it easy to check if all code were tested and what tests were still missing if the coverage was not yet sufficient. Depending on the difficulty of the tested class, the team used this tool to test each class with a coverage of at least 80%.

Rest:

- Git and Github
Git [7] is a very useful tool to share a project repository. The coach created a private repository via *Github* [8], the web application that represents the *Git* repository, to which the team members had access. Changes to the project could easily be merged into the main repository that the team shared. The team chose to use *Git* because its functionality to merge files that have been changed by multiple users is a lot easier to use than tools like *Subversion* or *Dropbox*. *Git* automatically merges different changes to one file and offers functionality to compare the changes when the file cannot be automatically merged. The team however did have some trouble using this tool where sometimes merging files would give a *dirty work tree* error, where the team lost a lot of time solving these problems.
- H2 Console
The application uses *H2* databases [9]. The route and strategy input files should be delivered as *H2* databases and the logs created by the application will be stored in a *H2* database. The team did some more research on databases and chose to use *H2* databases because of multiple reasons:
 - The databases are also supported in *Matlab*. The route and strategy inputs are created in *Matlab* and therefore it is important the data can be written to the file required by the application. Jan Hidders, professor of the Web and Database course, has also confirmed that *H2* is supported in *Matlab*.
 - *H2* databases are an embedded and can be used offline.
 - Furthermore, the resources that *H2* use take very low space and no installation is required to use it. No installation is required.
 - *H2* has a good support. On an online question of ours, we got a quick response from the main author of *H2* Database [10].
 - To view and manage the *H2* databases the *H2 Console* can be used. This tool is therefore very useful for Nuon Solar Team members that are less skilled in database queries. .
- VisualVM
VisualVM [11] is a tool to measure among others the amount of time that methods run. The team used this tool to analyse the start-up of the application, which takes about 20 seconds. The team used the tool to find out which method took a long time to run and tried to reduce the time these methods use. Using this tool the team found out that loading the *FXML* files took a long time, but could not be reduced.
- Wireshark
During the development phase the team encountered a bug where the connection with the Mobiboxx would fail (section 4.5). On our client's advice, the team used *Wireshark* [12] to capture all data send between the application and the Mobiboxx. Using *Wireshark* it was managed to fix the bug.

2.6.2 DESIGNER APPLICATIONS

- Scene Builder
Scene Builder [13] is a tool that is used a lot to visualize the *FXML* application of the application's GUI. More information can be found in the *Extension Manual*.
- Photoshop
Photoshop [14] was used to create the images and icons in the application. The team used template sets from existing icons together with some own adaptations and used *Photoshop* to give the icons the correct color and size.
- Visio and Astah
During the research phase the team designed the structure of the application and therefor created multiple diagrams of the application. The application of *Visio* [15] was used to create

the high level models like the *component model* and the *logical data flow diagram*. This tool was used to create good looking models, however it could only be used during a trial period lasting 60 days. The programming encountered a problem with this when the models were adapted during the development phase.

The class diagram was created using *Astah* [16]. This tool is able to create simple but clear UML diagrams. The main feature why the team chose to use *Astah* is the possibility to export the class diagrams to java files.

- Visual UML

During the research phase the team studied the source code of the previous application. Because the application was very poorly commented and not documented, the team tried to get a clearer overview of the application by converting the application to a class diagram. The tool *Visual UML* [17] was used for this purpose. The tool worked properly, however it printed the name of the tool over the created by the diagrams, which made the text hard to read.

2.6.3 PROJECT MANAGEMENT APPLICATIONS

- Dropbox

The team has chosen to use *Dropbox* [18] for storing all project files that are not part of the application itself. Files like reports, notes and tools like the *H2 console* were shared using *Dropbox*. The team chose to use *Dropbox* instead of *Git* for these files because *Dropbox* automatically uploads any changes to the files.

- Planbox and Trello

To manage the *SCRUM* sprints, the team started the project using the tool *Planbox* where items can be created. Each item can be divided in tasks and assigned to team members. For the research phase and most of the development phase the team used this tool to manage the sprints, but during the end of the development phase, the less important items were implemented, which resulted in a need for a better way to organize the backlog, which was a single list in *Planbox*.

The team started using the tool *Trello* [19] for this purpose. *Trello* is a very similar tool to *Planbox*, but allow for creating multiple lists and add items to these lists in which checklists could be added. *Trello* also was more usable in terms of moving items between lists.

3. RESEARCH PHASE

The research phase was the first phase of the project containing, in our case, three weeks. The main goals of this phase were to get a thorough understanding of the previous application, design the structure of the new application and research frameworks used for the new application. This phase was executed before the official start date of the project, because the calculated required time was not enough to meet the requirements of the project description.

3.1 PROBLEM DEFINITION

Based on the *Research Phase Report* () and our current documentation, the problem at the beginning of the project is as follows.

The Nuon Solar Team is a student team set up to show the world what is possible in the area of high-end technology and green energy. Therefore, every two years the Solar Team builds an improved solar car, the Nuna. With this solar car the team competes in the World Solar Challenge in Australia. Starting in 2001, the team won four times in a row. Their latest victory was achieved in 2013 with the Nuna 7 Solar Car.

Part of the success can be traced back to the concept of a chase car called 'Mission Control', which follows the Nuna. In the chase car, a laptop application receives all kind of telemetry and weather information from the Nuna car and several other sensors. This way the students can give the driver advice and anticipate their strategy.

This year, the team will compete in the Sasol Solar challenge in South Africa for the first time. This is done as an experiment to improve problems of the Nuna 7 in order to build a perfect Nuna 8. However, the application used in the Nuna 7 chase car also showed some problems.

The problem of the *Nuon Solar Team* that the programming team was assigned with was to create a telemetry application for Nuna's chase car. The application would have to be able to connect to the computer on board of Nuna, the Mobiboxx (the device inside Nuna that collects measurements and streams the data), and sensors in the chase car. The application would display the received data in a clearly structured GUI. The *Nuon Solar Team* already owns an application which complies with these requirements, but it was very hard to extend the program because most items were hardcoded, not documented and written in an old programming language, *Visual Basic 6*. Therefore only the developer of this application could change the application. Therefore a new application was necessary which was easily extendable.

Besides the product, the team will also deliver documentation. This documentation includes the *Research Phase Report* (), the *Plan of Action* (), the *User Manual*, the *Extension Manual*, the *Technical Manual* and this report. Additionally, the team's Final Presentation will give every stakeholder and an additional Bachelor Project course leader the opportunity to see the team's final product and review its quality.

The *Research Phase Report* will be the constructed report at the end of the research phase. It will explain our research and the corresponding results. The research phase report will be mostly used by the software team itself as the outcomes will be needed during the development of the product.

The *Plan of Action* document contains information about the goal of the project and how the team will achieve this goal. The document contains information about among others the project methodology and the deliverables.

The manuals are written to support the application for the users. The *User Manual* contains support for new users. It will explain how the application can be used and what functionalities are offered. The *extension manual* contains information about how to change the application. One of the requirements of the application was that the application is easily adaptable to changes to for instance the solar car. This manual will explain to the user how the user can adapt the application. The *Technical Manual* is meant for future programmers that will work on the application. It contains explanation about the implementation of the application.

3.2 PROBLEM ANALYSIS

During the research phase a list of requirements was created, clearly stating the requirements of the problem definition (section 3.1). This list is extended with requirement for the machine on which the application runs. The complete list of requirements for the application and the machine on which the application runs as created in the research phase are listed below. At the end of the project this list has been reviewed again to create an acceptance test (section 7.2)

3.2.1 USER REQUIREMENTS

The users of this application are members of the Nuon Solar Car strategy team. The user has a thorough understanding of the strategy and the data displayed in the application. Therefore the application does not display explanations of the displayed data. The user does not necessarily have an understanding of how the application works. Therefore the application runs flawlessly.

3.2.2 SYSTEM REQUIREMENTS

- Application runs on Windows version 7 or 8
- Java version 7 has to be installed on the machine where the application is running
- The machine on which the application is running needs to have the possibility to use a Wi-Fi connection
- The application could be easily controlled on touch screens
- When the application is not connected to the Mobiboxx, but it is connected to the sensors of the chase vehicle, the application will show estimates for the current power in and power out
- Application uses one connection to the Mobiboxx. One server application is connected to the Mobiboxx. Multiple client applications can be connected to the server application
- The application is available offline
- The properties of the solar car can be adapted in a config file. These properties are:
 - Car layout (images)
 - Battery package (layout, number of modules, number of cells, number of temperature sensors, battery state of charge-temperature curve)
 - MTTPs (number of MTTPs, position)
 - Model of power in
 - Model of power out
 - Mass of the car
 - Route (coordinates and locations of control stops)
 - Possibly other factors

3.2.3 FEATURES

To create a list of features we use the MoSCoW method. This means we divide all features in the following three categories: must have, should have and could have. In the 'must have' category you can find the features that definitely have to be in the final product. The 'should have' category contains the features that have to be in the final product, but not critical. The 'could have' category contains the features that are nice to have, but not necessary. The 'would have' category contains features that are unlikely to be implemented but that might come in handy for the next programming team.

3.2.3.1 MUST HAVE

This paragraph contains the features available in the GUI. The features are divided into the different components of the GUI.

General

- The following features are always visible: travelled distance, distance to finish, distance to the next control stop, current time
- All data is logged

Overview

Speed:

- Actual speed (km/h). When clicked: speed of motor, GPS of Nuna and GPS of MC (Mission Control, the chase car)
- Target speed (km/h). Can be changed on click
- Remote cruise control (on/off)
- Speed limit (km/h)
- Average speed over the last fifteen minutes (km/h). When clicked: speed of motor, GPS of Nuna and GPS of MC
- Expected speed in the strategy (km/h)

Battery state:

- Actual battery state (Wh). When clicked: CMS, BMS, model

- Battery power (W). When clicked: CMS, BMS, model
- Expected battery state in the strategy (Wh)

Power in:

- Actual power in (W). When clicked: CMS, BMS, model
- Average power in over the last fifteen minutes (W). When clicked: CMS, BMS, model
- Expected power in in the strategy(W)

Power out:

- Actual power out (W). When clicked: CMS, BMS, model
- Average power out over the last fifteen minutes (W). When clicked: CMS, BMS, model
- Expected power out in the strategy(W)

Connection state:

- State of connection with the Mobiboxx
- Message bar

Meteo

Shortwave radiation:

- Actual radiation (W/m²)
- Cloud fraction (%)
- Amount of radiation with clear sky (W/m²)
- Maximum (W/m² and %)
- Minimum (W/m² and %)
- Visual representation (bar) of fraction deviation
- Average radiation over the last fifteen minutes (W/m²)
- Expected radiation in the strategy (W/m²)

Effective wind:

- Visual representation (compass) of the direction of the solar car and the direction and speed of the effective wind
- Actual speed (km/h)
- Average speed over the last fifteen minutes (km/h). When hovered over: visual representation in the compass
- Expected speed in the strategy (km/h). When hovered over: visual representation in the compass

Wind:

- Visual representation (compass) of the direction of the solar car and the direction and speed of the wind
- Actual speed (km/h)
- Average speed over the last fifteen minutes (km/h). When hovered over: visual representation in the compass
- Expected speed in the strategy (km/h). When hovered over: visual representation in the compass

Temperature:

- Temperature (°C)
- Humidity (%)

Battery

Battery statistics:

- Battery state of charge (V). When clicked: CMS, BMS and model
- Largest amount possible (V)
- Maximum amount (V)
- Average amount (V)
- Minimum amount (V)
- Maximum temperature (°C)

- Average temperature (°C)
- Minimum temperature (°C)

Battery status:

- Amount per battery package (V). When clicked: temperature (°C) in the appropriate colour
- Per module:
 - Module status (error message/warning) in the appropriate colour
 - Current (A)

Power flow

- Visual representation of the solar car (top view), both the outside and inside
- Visual representation of the power flow between the MPPTs, battery and the motor
- Amount per MPPT (W). When clicked: voltage (V) and current (A)
- Battery:
 - Visual representation of the amount left in the battery
 - Amount (W). When clicked: voltage (V) and current (A)
 - Maximum voltage (V)
 - Average voltage (V)
 - Minimum voltage (V)
- Current per CMS (A) of the MPPTs, battery and motor
- Amount in the motor (W). When clicked: voltage (V) and current (A)

Status/temperature

- Visual representation of the solar car (top view), both the outside and inside
- Status per MPPT (error message/warning) in the appropriate colour
- Battery:
 - Maximum temperature (°C) in the appropriate colour
 - Average temperature (°C) in the appropriate colour
 - Minimum temperature (°C) in the appropriate colour
 - Status (error message/warning) in the appropriate colour
- Motor:
 - Temperature of the motor(°C) in the appropriate colour
 - Temperature of the air inlet (°C) in the appropriate colour
 - Temperature of the heat sink (°C) in the appropriate colour
 - Status (error message/warning) in the appropriate colour

Route

- Visual representation (map)
- Current position
- Route including control stops
- Zoom in/zoom out/auto zoom

Strategy

- Diagrams with the following data visualized as a function of time or distance:
 - Battery state of charge: CMS, BMS, model
 - Power in: CMS, MPPT, model (pyranometer)
 - Power out: CMS, motor, model (windmeter)
 - Speed: motor, GPS Nuna, GPS MC, target speed, speed limit
 - Effective wind speed: windmeter
 - Cloud fraction: pyranometer
- Visual comparison with the predicted strategy values
- Diagrams are separated in two parts: one zoomed in part for a period of one hour or one hundred kilometres and one part for the whole day or 700 kilometres
- Zoom settings: auto zoom, manual and default settings

Meteo (weather maps)

- Display of generated weather maps:

- Temperature
- Humidity
- Pressure
- Wind speed
- Display of layers on the basic map:
 - Low clouds
 - Mid clouds
 - High clouds
 - Precipitation
 - Wind stream lines
- Automated detection of available weather models and available runtimes
- Display of satellite images: visible/infrared/radiation prediction
- Animated weather charts

3.2.3.2 SHOULD HAVE

- The application is usable in multiple formats

3.2.3.3 COULD HAVE

- Auto speed adaptations based on strategy file (on/off) in mission control panel
- Advised speed (km/h) in mission control panel
- MT-SAT prediction of radiation (W/m²) in meteo panel
- In case of an error message or warning, the application will notify the user and will create a shortcut to the status/temperature panel
- A scroll function for the map
- Satellite panel with
 - Diagrams of the measured solar radiation, wind speed, wind angle versus predicted values of models and runtimes
 - Diagrams of predicted values based on the most recent strategy
 - Real time verification of the weather models including statistics (mean deviation, standard deviation)

3.2.3.4 WOULD HAVE

- Instead of an adaptable config file for the properties of the solar car, a set-up file which automatically saves the car, weather and country preferences

3.3 SYSTEM DESIGN AND STRUCTURE

After all the requirements were listed for the application, the team started with designing the structure of the application. Trying to design the structure of the application similar to the previous application, but in a way that the application is easily extendable.

First, the team created a component diagram (Figure 1) to create a high-level overview of the application and the different responsibilities of the application. The sensors, both in the Nuna and in the chase car, are external components. The server and client components are internal components as it are parts of the application. The subcomponents in the client and server show a high level impression of these internal components.

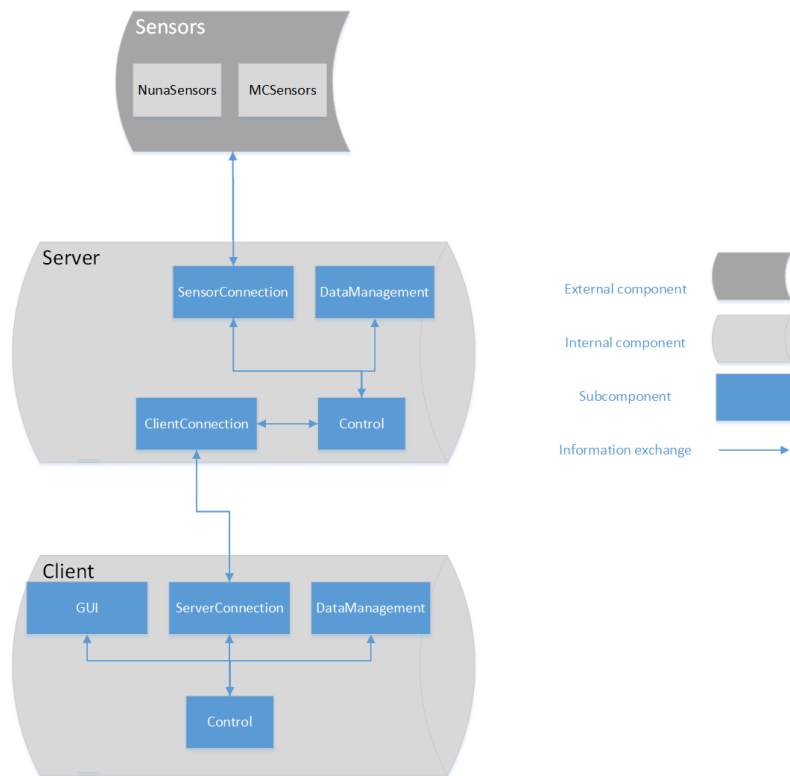


Figure 1: Component diagram before development phase

Another diagram is created to model the dataflow between the components in the application, the *logical data flow diagram* (Figure 2). The application contains three major data flows. The first green data flow is requesting sensor updates. The data flow originates in the *Controller* component, which sends a request to the *DataManagement* component to create the byte array message that should be sent to the Mobiboxx and the request to read data from the chase car sensors. The *SensorConnection* component will send the request to the sensors.

The second data flow is the black data flow from the sensors to the GUI. The sensors will send data to the application, which is received by the *SensorConnection* component. This component will send the received byte array to the *DataManagement* to be converted to a Java object containing the data. The Java object is then sent to the *Controller* component which passes it to the *ClientConnection* component to convert it to send it to all clients. The *ServerConnection* component will receive the Java object from the server and convert it to a Java object and pass it to the *DataManagement* for storage. The *DataManagement* component will send a notification to the *Controller*, which will pass the notification to the *GUI* component. The *GUI* component will then receive the Java objects from the *DataManagement* component to update the GUI with the data from the Java objects.

The third data flow is the GUI to sensors component. This data flow is used for the chat message and remote cruise control functionality. The user will execute a GUI action which results. The *GUI* component will create a Java object containing the user action. This Java object is passed to the *ServerConnection* component by respectively the *Controller* component and the *DataManagement* component. The *ServerComponent* will convert the Java object to a byte array and send it to the server. The *ClientConnection* component of the server will receive the byte array, convert it to a Java object and pass it to the *Controller*, which will pass it to the *DataManagement* component. This component will convert the data from the Java object into a byte array for the Mobiboxx conform the Mobiboxx protocol. This byte array is sent to the *SensorConnection* component, which will send it to the Mobiboxx.

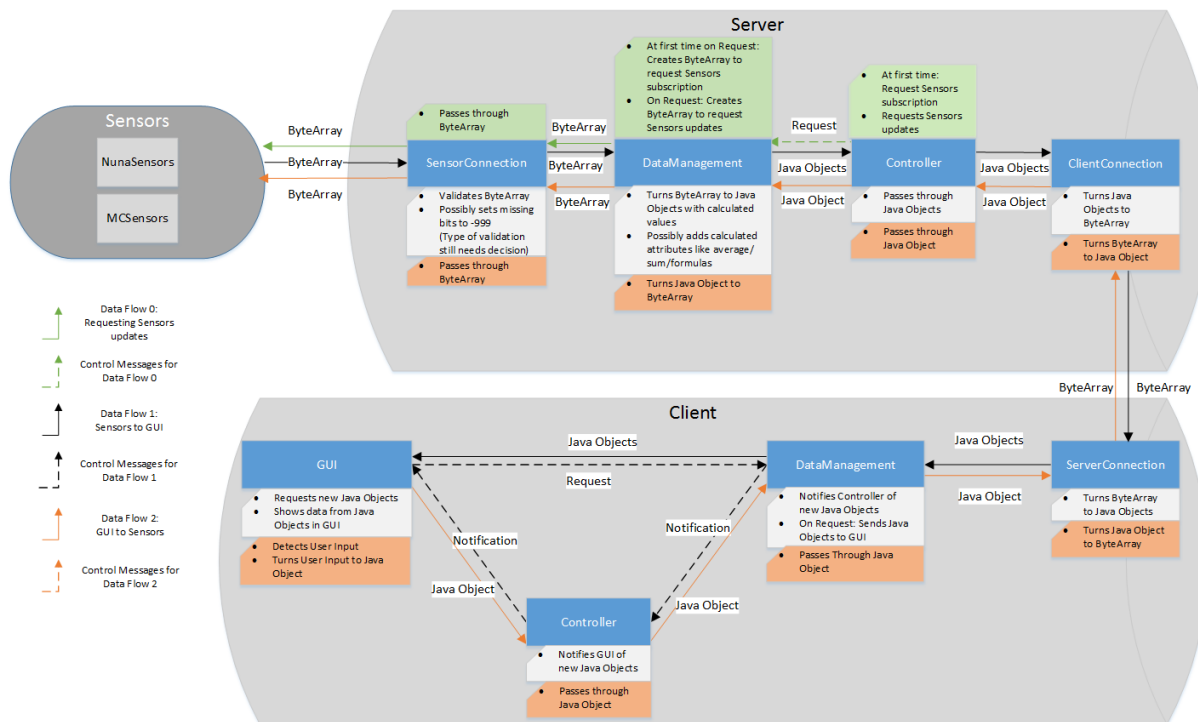


Figure 2: Logical dataflow diagram before development phase

After the different components of the application and the data flows between the components were designed, the application is modelled very specifically by creating a class diagram. The GUI part of the class diagram is incomplete because the team possessed insufficient knowledge about the GUI framework that would be used to design the classes of the GUI. The class diagram is split up in parts to fit in this document. The server part of the model can be found in Figure 3. The client part, except for the GUI, can be found in Figure 4. The GUI part is split up in multiple parts and can be found in Figure 5, Figure 6 and Figure 7.

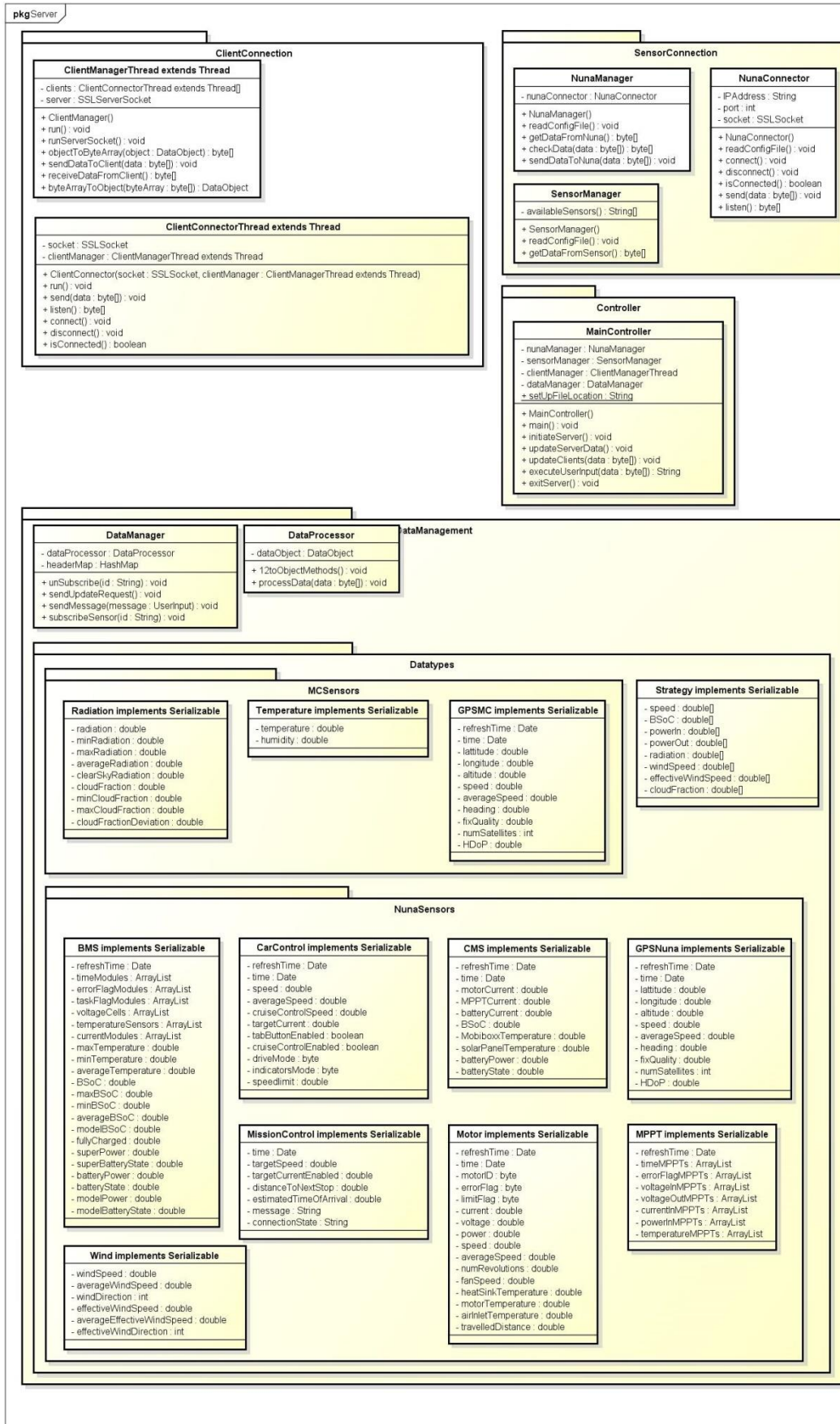


Figure 3: Class diagram of Server component before development phase

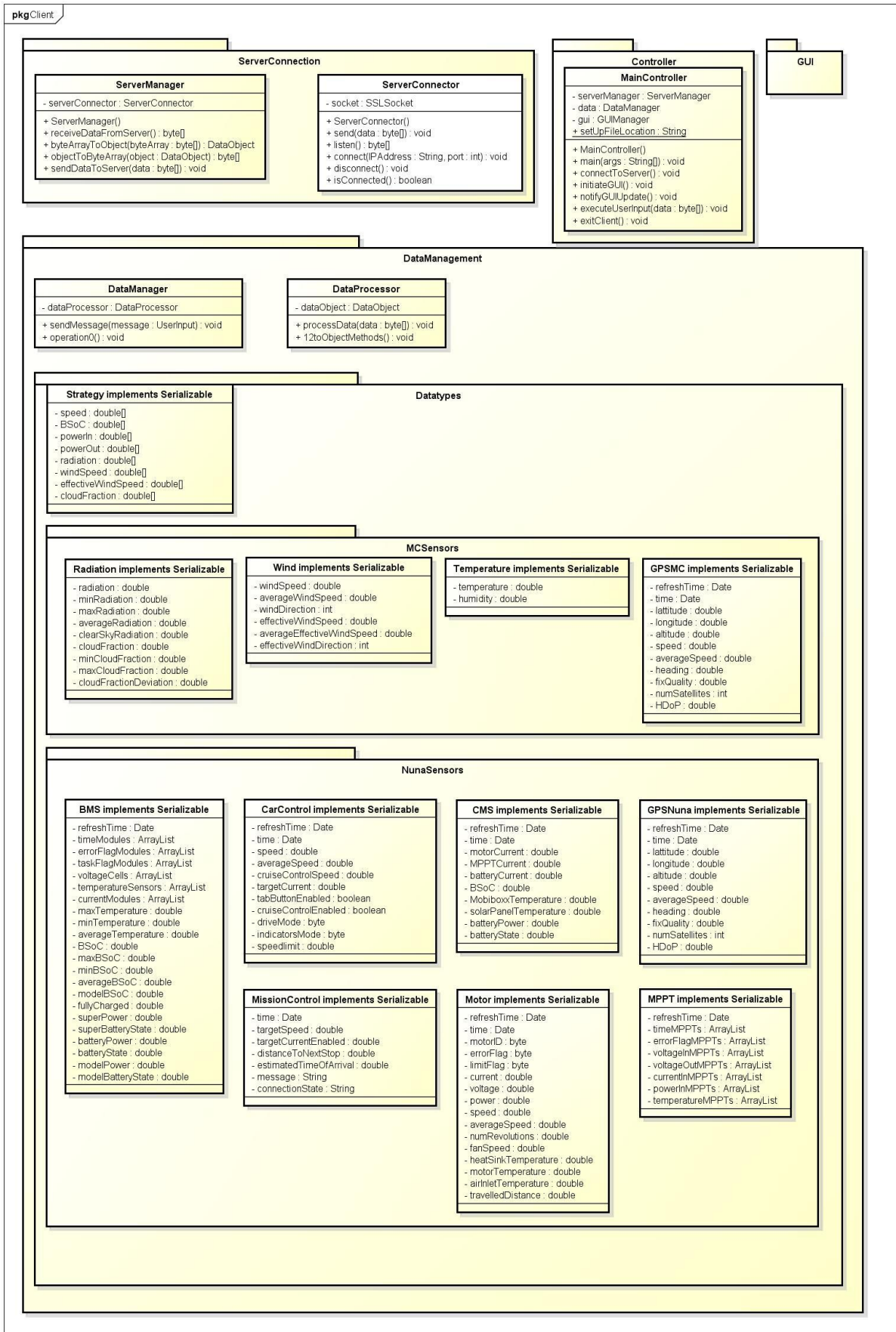


Figure 4: Class diagram of Client component before development phase

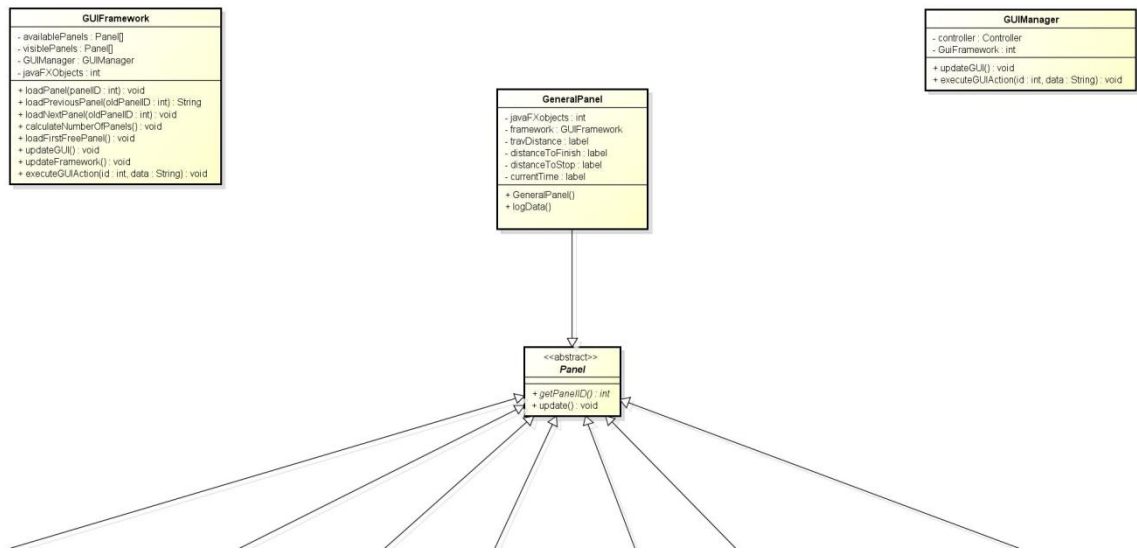


Figure 5: Class diagram of GUI component before development phase (1/3)

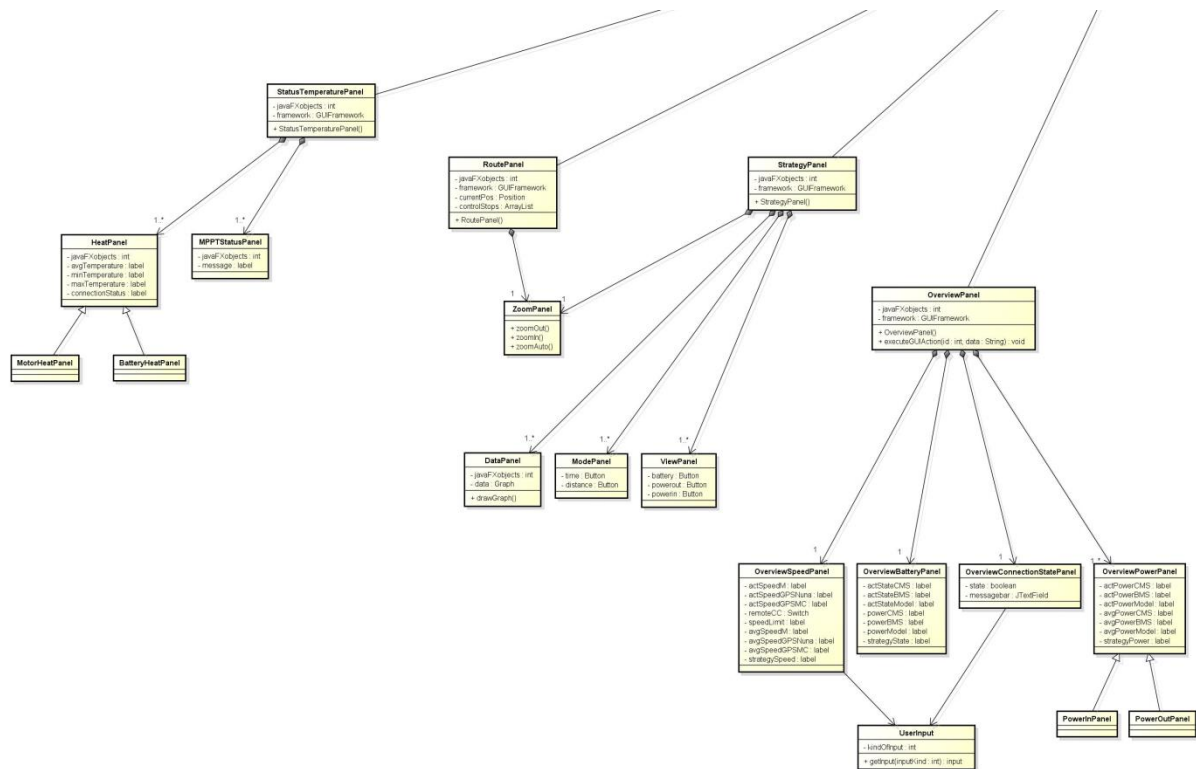


Figure 6: Class diagram of proposed GUI component before development phase (2/3)

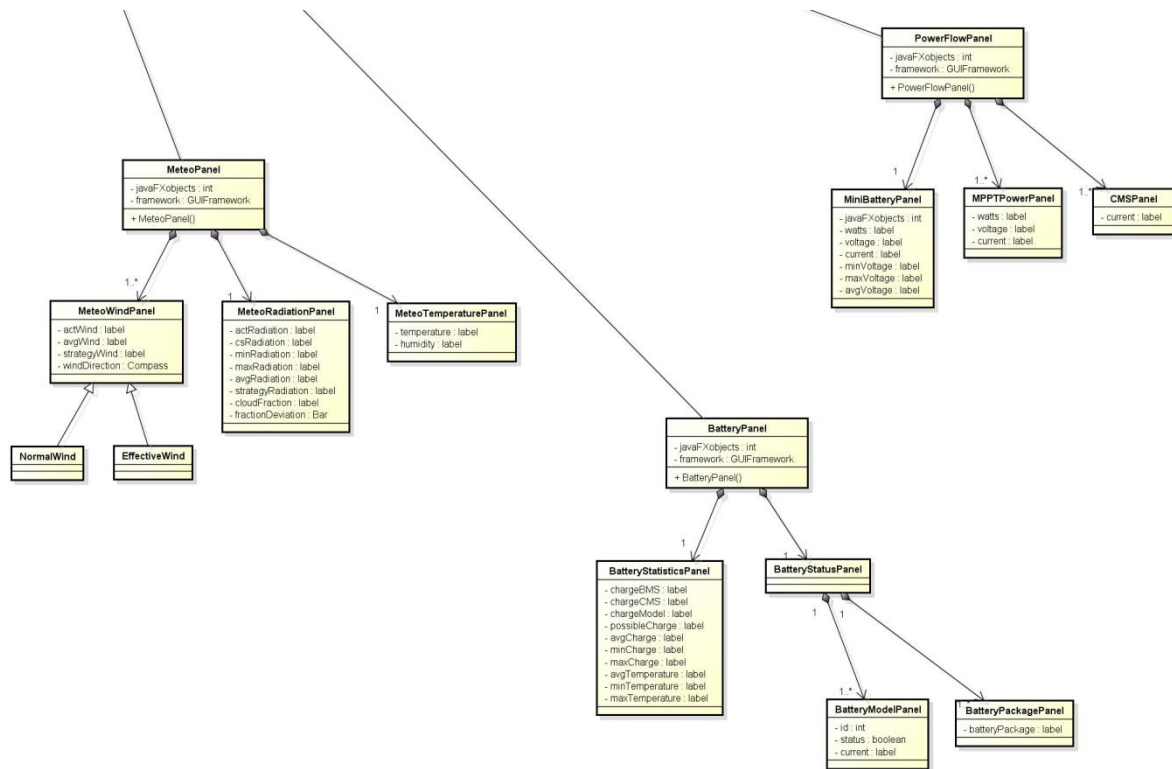


Figure 7: Class diagram of GUI component before development phase (3/3)

All modules in this paragraph are the models as they were created during the research phase. These modules are updated during the project and the up-to-date versions can be found in section 4.1. Over there, the up-to-date versions of the models are compared to the given models.

3.4 LANGUAGE AND FRAMEWORK RESEARCH

The team researched multiple programming languages for coding the application. After that several packages were investigated for goals as creating a GUI and setup connections to sensors and between applications.

3.4.1 PROGRAMMING LANGUAGE

When we looked at the old application and the *List of Requirements*, there were a few important criteria for the programming language of the new application:

- The programming language should support Windows 7 and 8. However, it would be very nice in regard to the application's robustness and extendibility if the language is easily adapted to other operating systems than Windows.
- The programming language should not require the user to install a lot of extra programs, packages, etcetera. This is especially true since the user does not necessarily have an understanding of how the application works.
- It should be easy for later software development teams to improve and/or extend the system.
- In order to suffice this condition, the language should be well documented. Moreover, it is preferable that the language is widely used, since this usually means that a lot of user tips, tutorials and examples are available on the Internet.
- It should be easy to implement a rich GUI
- This means that the language should have access to at least one framework or library focussed on the creation of GUI. The created GUIs should support multiple panels and be easily extendable, customizable and testable (section 3.4.2)
- It is preferable that the majority of the software development team is already familiar with the language because of the limited amount of time.

Three languages that would fit these requirements were discussed in the *Research Phase Report* (). The languages were chosen based on an initial pre-research on the Internet, own experiences and a company's presentation which was recommended by

one of the members of the Nuna Solar Team. Also, languages that are meant for Internet usage like *HTML*, *Javascript*, etcetera were disqualified as they could only be executed in browsers.

The outcome of these considerations was that Java would be a good language to help developing the *Mission Control* application. The reasons that were named are given below.

"Java is a widely used object-oriented programming language developed by Sun Microsystems. It is based on C++ but has a more extensive class library. Java is a very powerful language. Its main advantage is that it is written to be platform-independent. This means that Java could not only run on required platforms as Windows, but also on Mac OS. Also, the language is not completely dependent upon one IDE such as the LabVIEW or Visual Studio IDE, but can be executed in multiple IDEs. Together with its massive support and documentation, applications written in this language are the most 'open', robust and extendable of all three. The main requirement of our new application of easily improvable and extendable system could thus very likely be sufficed with Java.

For the users it is also easy to run a Java program. If not already present on their computer, they only need to install Java version 8.

As Java has access to a lot of internal and external GUI libraries, it will be very likely that there is at least one way to implement a rich GUI. Even graphical oriented implementing a GUI structure are available. A possible disadvantage lies in the fact that these libraries might not all be that clear, documented, and adapted to Java. Thus the very big advantage of Java, its platform independency, might also result in adequate but not great GUIs.

Lastly all five members of our software development team were very experienced in Java so the coding could be started right from the beginning of the development phase. This saves a lot of time and unnecessary helpless efforts.

Java suffices all of the proposed requirements in the most proper way. Although it does not differ that much from C#, personal preferences and experience form just the difference to decide to go for Java."

During the project the team has used Java and the team is very satisfied with the programming language. The decision taken by the team during the research phase has not been revised.

3.4.2 GUI FRAMEWORKS

The application will be used during races with the Nuna Solar Car. All information about the solar car and the sensors is displayed in the application so that the best strategy can be created during the race. Therefore it is important that all data is shown in a clear way so that during the race the strategists have a good overview of the data they want.

The team has researched multiple frameworks for a GUI in Java. In the *Research Phase Report* () it was stated that the framework should have:

- Multiple panels
- Easily to understand and to extend elements
- Customization options
- Support for testing
- Full screen support
- Scalable GUI support

The team researched Qt Jambi, Swing and JavaFX and the latter came out best because of these reasons given below.

"JavaFX is made by Java/Oracle to simplify making a GUI. Because Oracle, the owner of Java, made it, JavaFX is also well documented just like SWING. In contrast to SWING, this framework is not built-in for Java. Therefore it is slightly more difficult to implement this in a Java project. Some installations are required and imports of packages in the code are needed. This is a disadvantage to both the current development team and later development teams. But when this framework is set up, it makes it easier to create a GUI. The team experienced the framework as easy to use when trying to create a basic layout in an application and with addition of CSS it is also very easy to customise the components like buttons and texts.

The frameworks are compared and JavaFX is found to be the best option for this project. The most important part was the ability to customise the GUI easily. It is clear that the new application should look good, like the current application. Because JavaFX supports the use of CSS is JavaFX the best to work with. The greatest disadvantage is that it might be difficult to improve/extend the application for later development teams. Guidelines are going to be created to minimise this problem."

During the project the team has used JavaFX and the team is very satisfied with the GUI framework. The decision taken by the team during the research phase has not been revised.

3.4.3 CONNECTION FRAMEWORKS

The application gets all data of the Nuna Solar Car from a Wi-Fi connection with the Mobiboxx and the sensors. It is crucial for the application that all data from the Nuna Solar Car is displayed as soon as possible. Therefore it is important that the application can send and receive data over the connection fast and without problems. Since only one application (the server) can be connected to the Mobiboxx at the same time, the information needs to be distributed to all available running applications (the clients). In the *Research Phase Report* () it was stated that the network connection framework needs to be able to execute two tasks:

- Read data from the connection with the Mobiboxx
- Send the data to other available applications

The software development team has researched available frameworks in Java for this task. The researched frameworks were *Java.net* and *Javax.net*. In the end, there has been chosen for *Javax.net* because of the reasons stated below.

"The framework *Javax.net* is an extension of the *Java.net* framework. The *Javax.net* framework also offers the functionality of the *Socket* and the *ServerSocket* class. Additionally the *Javax.net* also offers the functionality to create secure sockets with the *SSLSocket* class and the *SSLServerSocket* class. Therefore the *Javax.net* is also a good solution as a real-time connection framework.

Javax.net is the best option for the real-time connection. This package offers the functionality required for the connections and additionally offers secure connections. Therefore the software development team has chosen to use the *Javax.net* framework for the real-time connections."

However, during the development phase there was an important change for the real-time connection framework back to *Java.net*. This was done because the client did not require the extra security offered by the *Javax.net* package. Because of this reason and the reason that *Javax.net* was found slightly harder to use as it requires proper security settings for the connection, the team decided to use the *Java.net* package instead.

4. DEVELOPMENT PHASE

The development phase was the most important phase of the project. In this phase the product was designed, created and tested. In this section the most important outcomes of this phase are given. These are our implementations (section 4.1), our approach to make the application extensible and changeable (section 4.2), our implemented extra functionalities (section 4.3), our test approach (section 4.4) and our challenges (section 4.5).

4.1 IMPLEMENTATIONS

In the research phase the application was designed at high level. The design was documented in some models which can be found in section 3.3. It is interesting to view and compare the final product that was implemented in the developing phase as some structural changes were present.

The most important change is that the sensor data updates are not push-driven anymore, but pull-driven. The original idea was to update the GUI every moment when new information is received. Because the update operations were slower and needed more CPU than originally thought, is this strategy changed so that the GUI now updates every second.

Also, as the classes were implemented, there were a few changes in the GUI component (which became the GUI package). As the proposed diagram for the GUI (Figure 5, Figure 6, Figure 7) was still very abstract and unclear, the packages and classes in the GUI have become more suited for JavaFX elements and better structured. The final design is modelled again and is explained below.

UML models are again used to describe the implementation of the application. The component diagram (Figure 8) shows that from our original high level application outline, nothing has changed.

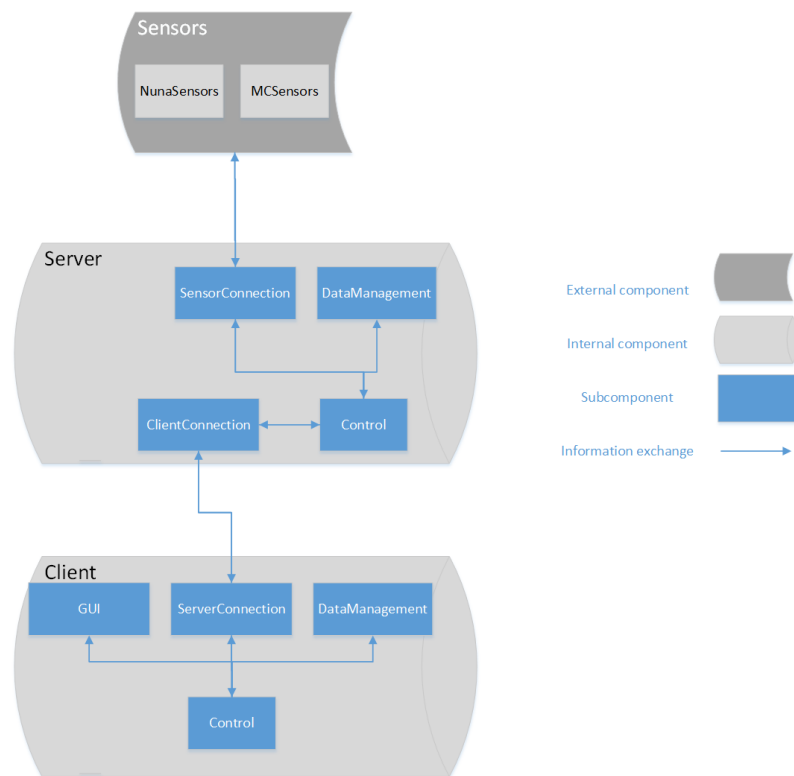


Figure 8: Component diagram after development phase

The client and server components could now be described more in detail. A complete class diagram of a big application becomes very big and unreadable, therefore a high level class diagram is made (Figure 9). In this diagram are only the most important classes shown, with the most important connections. Connections between classes in this diagram mean that the classes exchange information. The classes are grouped as in the code by packages.

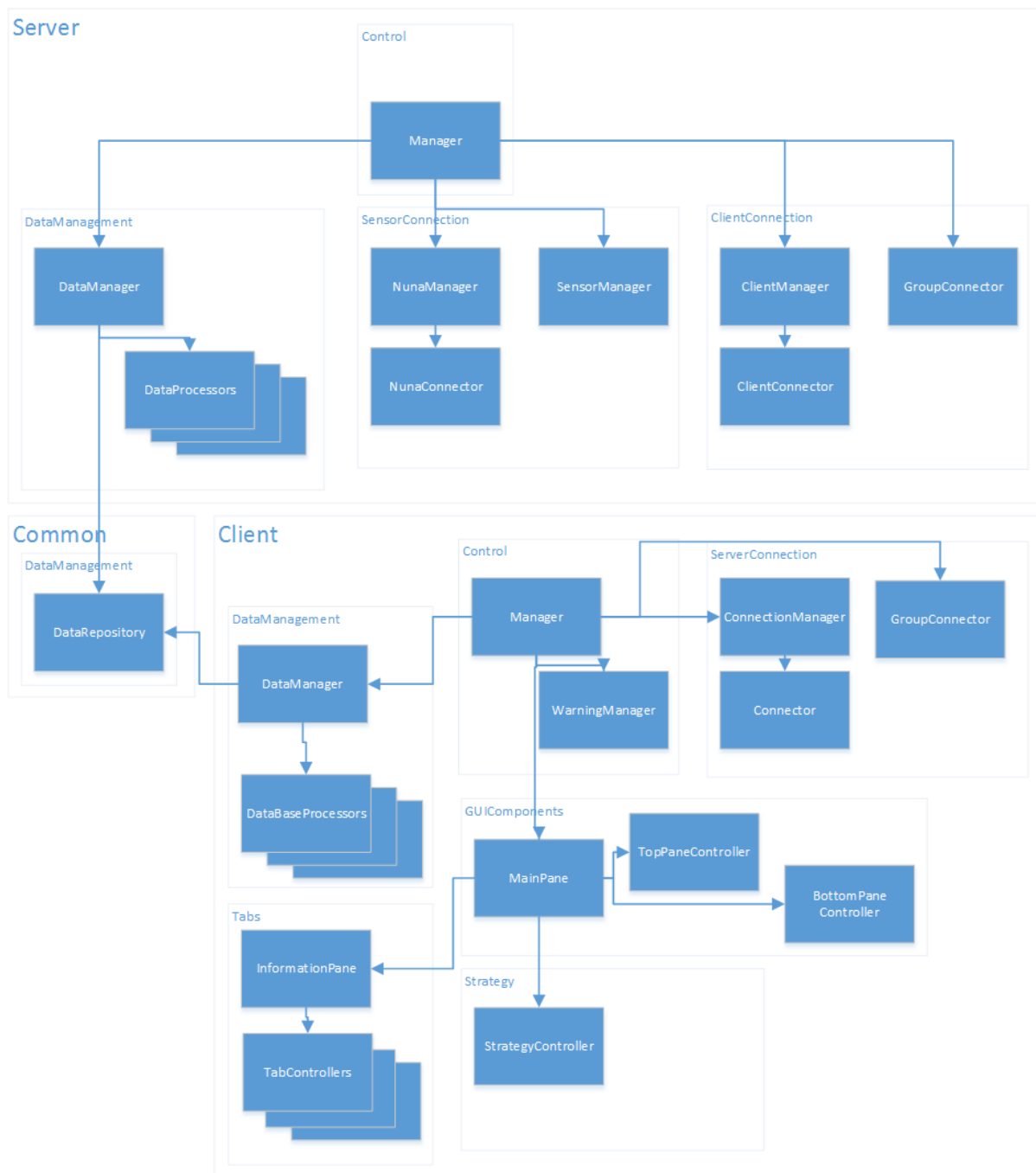


Figure 9: Lightweight class diagram after development phase

The server has three main tasks:

- Setting up connections to sensors
- Setting up connections to clients
- Storing data

These tasks are handled in separate packages: *sensorconnection*, *clientconnection* and *datamanagement*. These components of the server are controlled by the *Manager* class. The *Manager* delegates tasks and lets the components work together.

The way data from the sensors is stored, is also used in the client application. Therefore this part of the application is in a package called *common* and is used in both the client and server application.

Like the server, the client application has *Manager*. This class lets the subcomponents work together, which are doing the main tasks. The tasks of the client are:

- Connecting to the server
- Storing data

- Displaying the sensor data in a GUI

These tasks are distributed over separate packages. Connecting to the server happens in the *serverconnection* and storing data happens in the *datamanagement* package. Displaying the sensor data in a GUI is handled by many classes, divided over the *guicomponents*, *tabs* and *strategy* packages. The *guicomponents* package contains classes for the main GUI components like the top pane and the bottom pane. The *tabs* package contains classes for the detailed tabs like the tabs for the route, meteo and others. The *strategy* package contains all classes for the strategy part of the application.

The dataflow is another part of the application that helps to understand the application structure. The most important data flows are visualized in the next diagram (Figure 10).

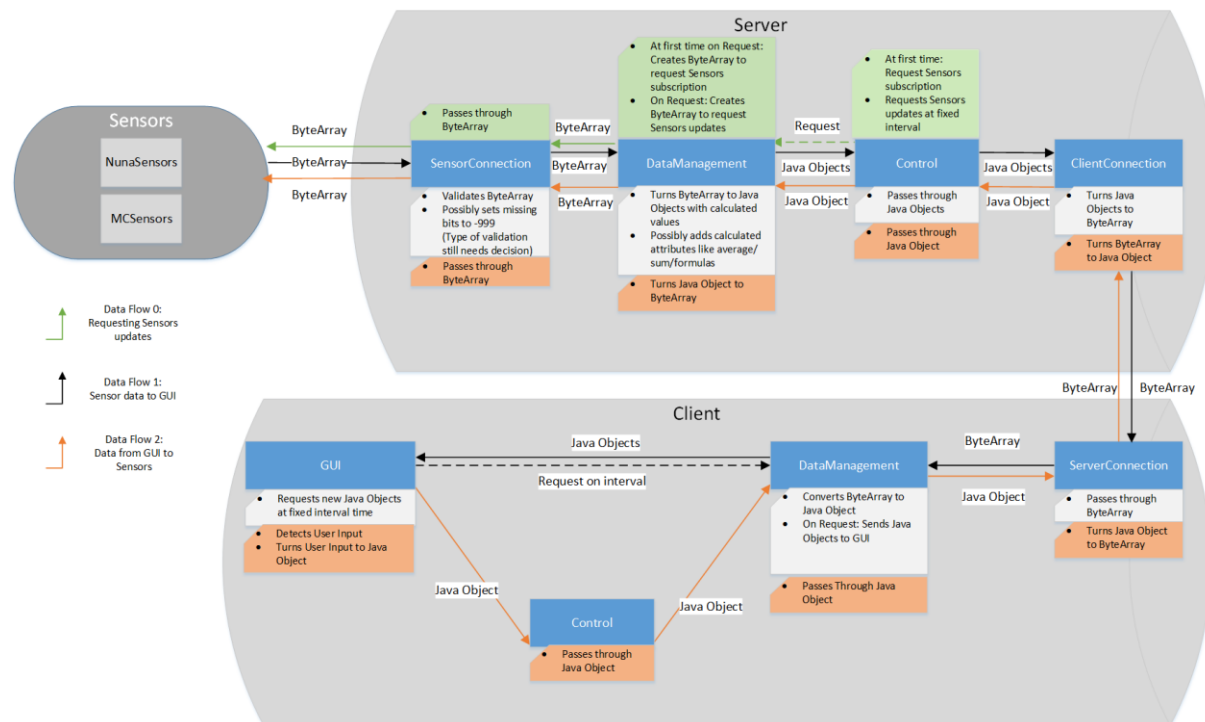


Figure 10: Logical dataflow diagram after development phase

As is visible, the GUI will now not receive data in a push-driven way anymore, but pull-driven. Furthermore there were no big changes in the main data flows that were already described (Figure 2).

4.2 EXTENSIBILITY OF IMPLEMENTATIONS

One of the main goals of the bachelor project was to deliver an application which was extendable. The extendability of the application is an important thing, especially for the future developers. Our application is extendable in many ways. In general, it tries to give some power of the programmer to the user in a way that is understandable for them as well. The following list sums up the most important items in our application that are very easy to change or extend:

- System configuration
- Structure of the application
- Layout of the application
- Strategy functionality of the application

System configuration

The system configuration is easily extendable by using a configuration file. The configuration file is a file in which the settings of the application, the specification of the solar car and the race can be modified. This file can be found in an external folder so that the user can easily modify it. The configuration file contains many items that can easily be adapted or changed. The following XML blocks can be found in the file that was delivered at the application's release:

Block	Description
<Data>	Root, contains all other blocks
<Program>	Settings for application
<Mobiboxx>	Settings for connection to Mobiboxx
<IPAddress>	IP Address used by the connection
<Port>	Port number used for the connection
<Server>	Settings for the connection to the server application
<Port>	Port number of the server part of the application
<GroupConnection>	Settings for group connection where the server broadcasts its IP Address
<IPAddress>	IP Address used for the group connection
<Port>	Port number used for the group connection
<Headers>	Contains headers for the communication with Mobiboxx
<Subscribe>	Header for Nuna sensor subscribe message
<Unsubscribe>	Header for Nuna sensor unsubscribe message
<DataRequest>	Header for Nuna sensor request data message
<Data>	Header of data messages that are received from Mobiboxx
<MissionControl>	Header of MissionControl message that is sent to Mobiboxx
<SensorsUsed>	Settings for sensors that are used by application
<NunaSensors>	Settings for Nuna sensors that are used by application
<Sensor> - <...>	Value (true/false) indicating if application uses the sensor
<MissionControlSensors>	Settings for Mission Control sensors that are used by application
<Sensor> - <...>	Value (true/false) indicating if application uses sensor
<SensorConfiguration>	Configuration settings for sensors of chase car
<Sensor> - <...>	Connection settings for sensors of chase car
<Port>	COM-port on which sensor is connected
<BaudRate>	Baud rate of sensor
<TabsVisible>	Settings of tabs that are present
<Tab> - <...>	Value (true/false) indicating if tab is present in application
<CriticalValues>	Settings for values that can give warnings in application
<Element> - <...>	Settings for elements (warning category) that give warnings
<Value> - <...>	Value that can give a warning when too high or low
<LowerRed>	Value is always red beneath this number
<LowerYellow>	Value is between yellow-red + red warning flashes
<LowerGrey>	Value is between grey-yellow + yellow warning
<UpperGrey>	Value is grey thus okay + warning disappears/turns grey
<UpperYellow>	Value between yellow-grey + yellow warning
<UpperRed>	Value is between red-yellow + red warning flashes
<GreyDeviation>	If value differs this far from mean, slightly darker/lighter grey
<ResumeFlashing>	If value differs this far from old value, new warning is given
<ToTab>	Tab that is opened when warning item with value is clicked
<CalculationConstants>	Constants used for calculations
<PowerInModel>	Constants used for the power in model
<Constant> - <...>	Constant values used for the calculation of the power in model
<PowerOutModel>	Constants used for the power out model
<Constant> - <...>	Constant values used for the calculation of the power out model
<SolarCar>	Settings for solar car representation in application
<BatteryConfiguration>	Settings for configuration of battery
<Module>	Settings for module in batter
<Cell>	Settings for cell in module
<TemperatureSensor>	Sensor that helps calculating temperature in this cell
<FullyCharged>	BSoC when the battery is fully charged
<MPPTConfiguration>	Settings for configuration of MPPTs
<MPPT>	Settings for one MPPT
<PercentX>	Percentage of visual x-coordinate of MPPT in application
<PercentY>	Percentage of visual y-coordinate of MPPT in application
<PercentXStatus>	Percentage of visual x-coordinate of MPPT in Status tab
<MassOfCar>	Mass of solar car
<Race>	Settings for race representation in application
<RacingDays>	Settings for days that are used for race
<Day>	Settings for one day that is used for race
<Date>	Date for this day
<StartTime>	Start time of this day
<EndTime>	End time of this day

Structure of the application

The structure of the application can also easily be modified. With structure we mean the different GUI elements in the application, e.g. the tabs or the ordering of values. An external program called the *Scene Builder* can be used to change the GUI elements, and therefore the structure of the application. The *Scene Builder* makes it easy to make the design of a GUI without having any (or little) background knowledge in programming. More details about JavaFX can be found in the *Research Phase Report*.

Layout of the application

The layout of the application can also easily be changed because the style sheet (GUI.css) is an external resource. Things like the color, shadow and border of GUI elements can easily be changed by adapting this file.

Strategy functionality of the application

There is functionality of the application that can be extended by the user. The strategy contains multiple configurations, to make plotting of race information easier for the user. A user can also help himself by adding configurations in a very easy way. The configurations, which are xml files, can all be found in the external resources folder. The user can add a file and thus a new configuration, making the application do more than it normally does.

Furthermore, our product clarifies the extendibility of the application, and shows what each component and each class can do. Adding more features can be done by the programmer and, to some extent, by the user, as mentioned above. It can also be used for different races and different racing types like races with loops or races without.

4.3 EXTRA FUNCTIONALITIES

Some extra functionalities, which the Nuon solar team did not ask for have been implemented as well. The client is happy about these extra functionalities and they certainly help in the overall experience of the interaction with the application. The following list shows the extra functionalities that have been added:

- Starting and closing server from GUI
- The bottom progress pane
- Warning functionality
- Resizable screen
- History tab

Starting and closing the server from the GUI

The starting and closing of the server from the GUI is extra functionality. The application did not have this functionality, and is a feature that is added by us. It comes in very handy because the user only needs to start the GUI and click a button. The client found it a good feature, something that we were very happy with.

The bottom progress pane

Then there is the bottom progress pane, in which the user can see the progress of the race. He can see a car and its travelled distance, which gives a good overview on how far the finish is. This feature came as a surprise for the client, but also for the Nuon solar team, as they did not expect it. They found that a little car driving and showing the travelled distance was a good idea.

Warning functionality

We have also added warning functionality in the top pane. If attention to specific components like the battery or solar panels is needed, it will be shown in a warning list. Also, after clicking on the desired warning, the user will be guided to the corresponding panel.

Resizable screen

The resizing of the screen also belongs to the added extra functionalities. Resizing the screen comes in handy for when the user wants a smaller screen, but also to move the screen to a second monitor. The screen can be resized by width or by height. The user can also choose to set the application to full screen mode.

History tab

The history pane is functionality which has not been required by the client. We have chosen to implement the history pane ourselves. The history shows the logs of the sensors in a clear way. The client used to review the logs by using a program like notepad, which is not user friendly. There were multiple log files (text files) that needed to be opened separately before a user could review them. With the history pane, these problems disappear. From the history pane, the user can choose which type of log to show, with a clearer overview than with a program like notepad. All logs are now saved into a database, having one place for saving all logs. The client's response to the history pane was very positive, which we found satisfying as we wanted to improve the user experience.

More information about the functionality of the application can be found in section 6.2.

4.4 TESTING IN DEVELOPMENT PHASE

Another part of the development phase consisted of testing the implemented source code. There were several ways of testing:

- Unit testing
- Integration testing
- System (real life) testing

Unit tests were created for all implemented methods where possible in order to create an overall high percentage of test coverage. Integration testing was done to make sure that packages in the source code could not only function on their own, but also together with other packages and still works as expected. System tests involved two kinds of testing, namely static testing and testing days. Static testing was done when the Nuna stood still in the workshop in order to verify that data from the Nuna and Mission Control sensors was received. Testing days included testing with the Nuna and Mission Control driving, so representative data could be received and verified.

In the beginning there were some difficulties with the unit tests, especially with testing the connection between the server and client, the connection between the server and the Nuna, and testing the GUI. The main problem with creating tests for the connection was that an actual connection had to be set up in order to pass certain tests. For unit testing this was impossible because only integration testing would resolve these problems. Testing the GUI resulted in a much more basic problem, which was simply that there did not seem to be a sufficient package for testing JavaFX code. The major parts of the packages were abandoned projects or still only ideas. Eventually there was the TestFX package that could be used for proper GUI testing.

The first real life testing day was very important regarding the usability of the application. During the testing day it became clear that there were some significant bugs that could only be found until then, because an environment similar to a real race had to be created. The most important bug that was found was that when the connection with the Mobiboxx was lost, it seemed as if the connection was not re-established. After thoroughly revising the source code of the connection, it turned out that in fact reconnecting to the Mobiboxx occurred as desired. The problem was that the server did not subscribe again for the sensors of the Nuna, which is necessary for receiving any data of these sensors. Then there were some minor bugs, for instance the speed of the Nuna was displayed incorrect in the GUI, because the value was not sent in km/h as assumed but in m/s. By making a small change this was easily solved, but made a great difference for the end-users.

For every meaningful validation test, we have made test plan. We have included this test plan to this report ().

4.5 DEVELOPMENT CHALLENGES

In the development phase, we faced a lot of challenges. Below, we have tried to list the most important ones.

Challenge 1: Connection with hardware and sensors

The first challenge to mention is the connection between the application and the Mobiboxx. A standard setup for connecting with the Mobiboxx had to be implemented first. This connection uses the client server model, which had to be implemented in Java. Before the project we had never implemented and used a proper client and server model. It was quite a challenge to accomplish but we succeeded.

A lot in this field has been learned. For example, at one moment, after a big structure change, the team worked some days away from the Nuon Solar Team offices. After three days, when we returned, suddenly the Mobiboxx connection was not established anymore. After days and days of reverting code, trying to do every change manually, we managed to fix the problem with the help of the *Wireshark* program, recommended by our coach. This was certainly a situation where we have learned to directly test if everything works again, even when only structural and no functional changes have been made.

Also, we managed to work with telemetry sensors that actually require some knowledge in electrical engineering. We have never worked with them earlier and it took us more time than expected to even receive some data that was making any sense. However, we succeeded and learned a great deal from this.

Challenge 2: Storing all data efficiently

Our second challenge was efficiently storing all data. We have indeed learned to store this efficiently, allowing one repository in which all the java data objects are held. All objects are efficiently saved in this data repository and are only made once. These objects are also serialized and transferred between the client and server.

Challenge 3: Work with databases

During the project, we set another challenge for us, since we preferred database inputs as Strategy instead of the Text files. The database was used for logging and for processing some strategy information of the application. A course in our bachelor, Web and database systems, came in very handy. We already knew how to write SQL-queries.

For this objective we had to do some extra research into databases (section 2.6.1). The reason that we preferred databases is because we had good experiences with it, and we knew you could do some useful, fast operations with it. In short, these were our main reasons of choosing a database:

1. You can look up data from a database relatively quickly
2. You can relate data from two different tables together using JOINS
3. Your data has a built-in structure to it
4. Information of a given type is always stored only once
5. Databases are ACID [21]
6. Databases are fault-tolerant
7. Databases can handle very large data sets
8. Multiple users can use databases at the same time without corrupting the data
9. Databases scale well

Reasons based on [20]

The user is now able to retrieve specific information faster, especially since we also explained in the manuals how to retrieve specific information about database information.

Challenge 4: GUI development

The last challenge was to design the GUI. Designing a GUI was something that we have never done to this extent and certainly not a GUI that should include that much values. The key points here were to design a GUI that

- Contains all information
- Has data that can always be clearly seen
- Could be used by touch screen devices, without any mouse
- Has a clear overview for its users of where (semi) hidden data can be found
- Does not contain any unnecessary or duplicate information
- Is resizable
- Runs on multiple screen sizes

In short: we have tried to deal with these difficulties, another challenge that we have overcome. Using JavaFX was a great experience and we learned a lot from it. The creation of GUI elements in the Scene Builder, the possibility to set a controller for a GUI element and also editing the GUI from the Java code itself were learnings.

Before this project we only had a limited experience in using Swing, a build-in GUI library in Java. However, since this project, JavaFX has become our preferred way of designing a GUI with Java. This will be of good use for future Java projects.

5. CONCLUDING PHASE

5.1 TESTING IN CONCLUDING PHASE

In comparison with the development phase, testing in the concluding phase consisted more of system testing. This was to make sure the application worked as the client desired. There have been a few testing days where there was the possibility of testing the application as if in a real race. Not only was this a great experience, but also very important for fixing the last bugs and improving the functionality of the application.

In the concluding phase there was also a new kind of testing, namely the acceptance testing with the client. For this kind of testing, functionality of the application was presented and explained. Then the client had the possibility of making clear if everything worked as expected and perhaps give some suggestions about certain elements that could be improved or slightly changed. This way of testing was very convenient for making sure that the application met all the requirements.

5.2 SIG CODE EVALUATION

5.2.1 FIRST EVALUATION

After finishing most of the application, the application needed to be evaluated by the Software Improvement Group (SIG). The result of this evaluation was that the application scored generally well, with three out of five stars for maintainability. Different important points of the feedback are discussed, as well as our reflection on it. The first part of feedback:

"What is noticeable when looking at the code is that, in the file system, there is no clear component structure visible. This makes it difficult for a developer to have a general view of the functionality that the system offers. We strongly recommend dividing the code into different (functional) components to give an impression of the high level structure of the system."

To summarize it shortly, this feedback is about dividing the application into different components so that a developer can have a quick idea about the high level structure of the system. Our reflection on this feedback was that it was not clear enough. After asking a more clear explanation of the feedback, it seemed that the components were internally divided well, but need a better separation at a higher level (in the folder hierarchy). We believed that something went wrong with sending our source code to SIG, because we already had this separation. We agreed with SIG that both parties will keep an eye on the structure at the hand-in of the code for the second evaluation.

Secondly, SIG gave this feedback:

"The .java and .class files are contained in one folder. Normally, these files are distributed among separate directories (usually a source and bin folder) to prevent that compiled files pollute the source tree."

This feedback suggests that all the .java and .class files are mixed in one folder, which is not a very good design choice. However, we believed that something went wrong here as well, because the project is structured in such a way that this could not happen. When looking at the structure of the project, it can clearly be seen that this problem is not the case. Again, we agreed with SIG to email them on the second attempt of the code evaluation, to make sure that the problems do not occur twice.

The third part of feedback:

"Unit complexity represents the degree in which the code is, above average, complex. The splitting of this complex code into smaller pieces ensures that each component is easier to understand, easier to test and therefore easier to maintain. In your case, some methods are implemented in a complex way, while this could be done much easier. Take for example the function `BMSDataProcessor.getBMSErrorFlagModules()`. What you actually want it to do is to look up the correct error flags in a map with error codes and error messages. The key of the map is the error flag module, and the value is the text belonging to the key. Doing it this way you do not need to have a switch function."

Some parts of the source code were written in a complex way. SIG gave as an example the `getBMSErrorFlags()` function, which retrieves the current error flag of the Battery Management System. The implementation of this function was indeed a little complex. To solve this problem the advice SIG gave was followed up, which means that switch function was replaced by a map filled with all elements. We solved this problem for all similar complex functions to reduce the unit complexity.

Then there was also feedback about the module decoupling:

"Module coupling represents the amount of code that is frequently called. Code that is frequently called normally leads to a less stable system, because changes in this type of code can lead to changes in many different places. InformationPaneController is an example of a class that is frequently used. There are a number of methods in this class that you could move to a utility class, like the functions `stringValue ()` and `createDashes ()`. That way you will avoid that the InformationPaneController class will slowly become a god class."

Code that is often called can make a system less stable, because changes in these pieces of code can create a chain reaction of other changes. SIG pointed out that the `InformationPaneController` class is frequently used and could turn into a god class. The advice of SIG was followed up and thus the problem was solved by creating a utility class and moving all utility functions to this class.

The final part of feedback was:

"The presence of test code is promising. We hope that the volume of the test code grows with newly added functionality."

5.2.2 SECOND EVALUATION

After we adapted our code according to the recommendations of SIG and had extended our system for new functionalities, the code could be uploaded for the second SIG evaluation.

"In the second upload we can see that both the size of the system as well as the score for maintainability has risen. The component structure now has a clear subdivision in 'client', 'server' and 'common'. This makes it more clear. We noticed that the 'client' has a direct dependency of the 'server' (to start the server locally), it would be a good thing to document the reason for this."

"In the area of Unit Complexity we can see a rise of the score, the more complex methods are changed in a good way. Besides that we can also see that the recommendation about Module Coupling have been incorporated. The last thing that stands out is that the amount of test code has risen more than the amount of production code (18% versus 46%)."

"From these observations we can conclude that the recommendations of the previous evaluation have been incorporated in the development process. It is good to see that besides an improvement in the maintainability there still is a rise in the amount of test code."

We were very happy that the evaluation stated an improvement of our system. The most important outcome of this evaluation was that the maintainability has risen to four out of five stars. This means that our product actually is appropriate for further development by members of the Nuon Solar Team.

5.3 DOCUMENTATION OF FINAL PRODUCT

We have already explained how we have made sure that our code is extensible (4.2). However for this project, it was also a major requirement to make our product maintainable for our successors (3.1). This is why our application should be delivered in a way that it is understandable for any successor by documenting our work.

Defining what should be documented turned out more difficult than expected. Since our formal client was a member of the previous Nuon Solar Team and thus was not going to use our application, there were different stakeholders who will get involved into our product. Some successors should be able to fully understand our application, while others should only be able to use it. Thus we defined our successors in terms of the following three target groups:

- Current Nuon Solar Team members

- Members that simply want to use the application as soon as possible without configuring much.
- Future Nuon Solar Team members
Members that want to completely change the application configurations to their own wishes, or extend it.
- Future software development teams
People that want to completely change the application's code, structure and lower level elements.

For each target group, a manual has been written with exactly the contents that should concern them. All of these manuals will be included in the final documentation package that will be delivered next to the application code package. The manuals, sorted on their respectively mentioned target group include:

- *User Manual*
- *Extensibility Manual*
- *Technical Manual*

6. RESULTS

The result of the research phase, development phase and the concluding phase is the Mission Control application. As said the application comes together with the *User Manual*, *Extensibility Manual*, and *Technical Manual*.

6.1 PACKAGE CONTENTS

The application is provided in a specific package structure (Figure 11).

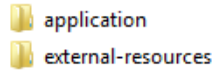


Figure 11: Package contents

The application is compiled as a runnable jar, *Mission Control.jar*, and can be found inside the *application* folder. The *external resources* folder contains files that are used by the application and can be configured to the user's wishes (

Figure 12).

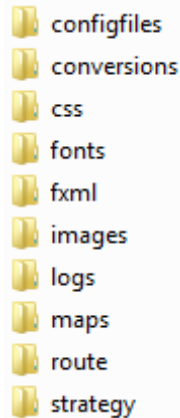


Figure 12: External-resources folder

6.2 LAYOUT OF THE APPLICATION

6.2.1 OVERVIEW

The application will first show a loading screen when started. Then the application will start in the sensor view (Figure 13). In the application there is the possibility of switching to the strategy view (Figure 14). An overview of the main elements is given below.



Figure 13: Overview of the application in sensor view

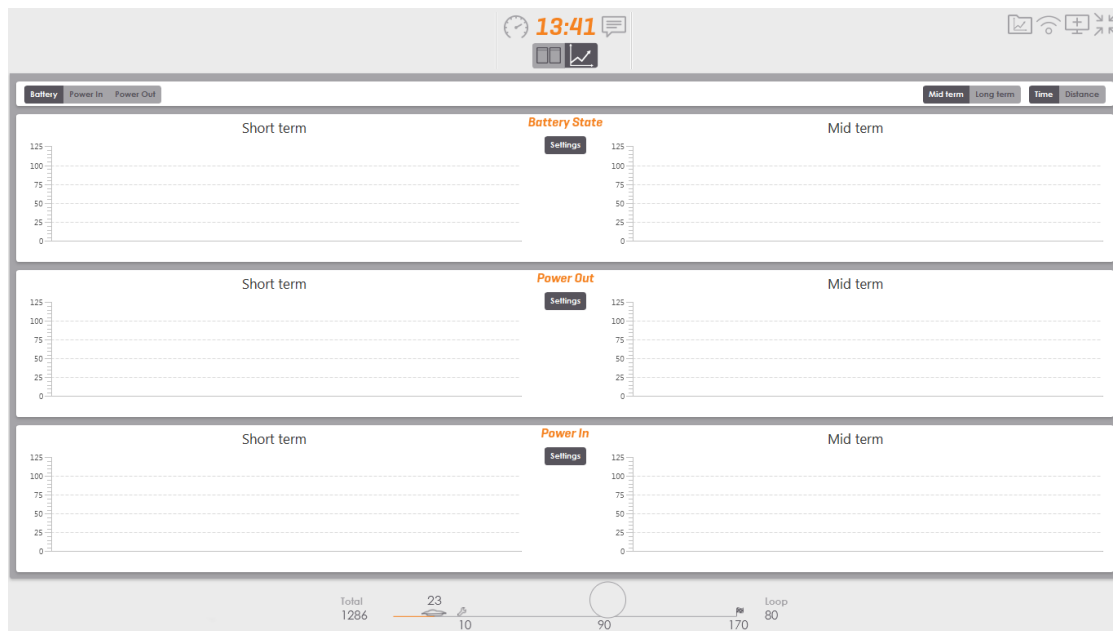


Figure 14: Overview of the application in strategy view

6.2.2 SENSOR VIEW

In the sensor view it is possible to switch between tabs in the right and left information pane. Once a tab is selected in one information pane, it cannot be selected in the other one. The tabs each serve a different purpose which could be found below (Table 3).

Sometimes, a resize event could reduce the number of information panes to one when the size of the screen has only room to display one tab. This way, the screen could be dragged to one of the sizes of the Windows desktop and there is room for more applications.

Tab icon	Name	Description
	Overview	Most important values of all components
	Battery	Values of the battery component
	Power Flow	Values of the power flow between the MPPTs, BMS, CMS and motor
	Status	Information about the connection with all components
	Meteo	Values about the weather conditions
	Route	Information about the route on a map
	History	Logs of a race can be viewed

Table 3: tab icons with names and description

In the Overview *tab* (Figure 15: Overview tab in the application) the most important information of all elements are shown. There are five blocks: Speed, Speed Guidelines, Battery, Power In and Power Out. The size of the values is determined by the importance of a value.

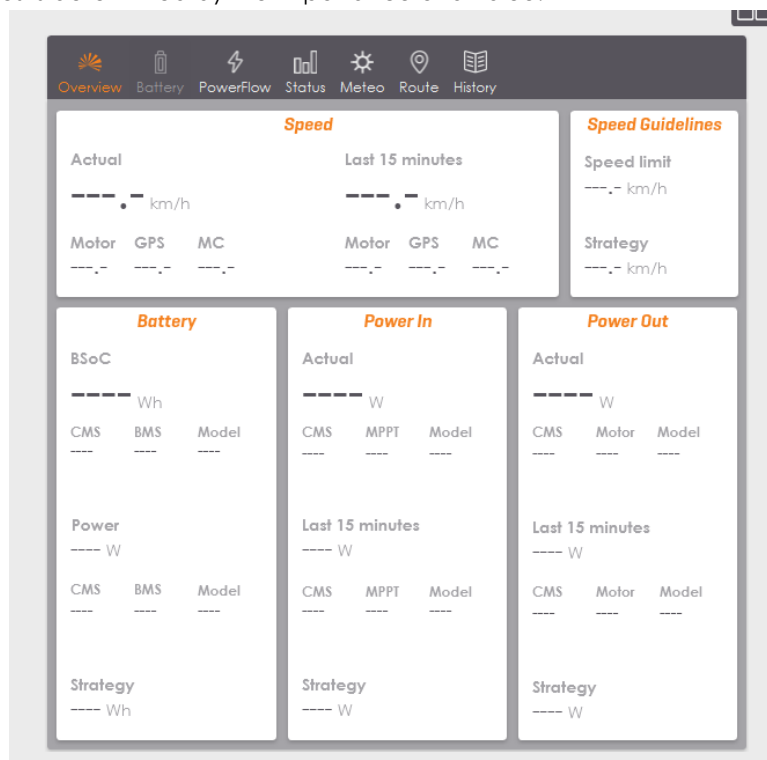


Figure 15: Overview tab in the application

In the *Battery tab* (Figure 16) all information about the battery and its cells is shown. There are four blocks: Battery State of Charge, Voltage, Temperature and Cells.

In the Cells block there are switches which represent the cells. Voltage is shown in each switch by default. This can be changed to temperature by clicking on the switch. The voltage/temperature button makes it possible to change the view of all switches at the same time.

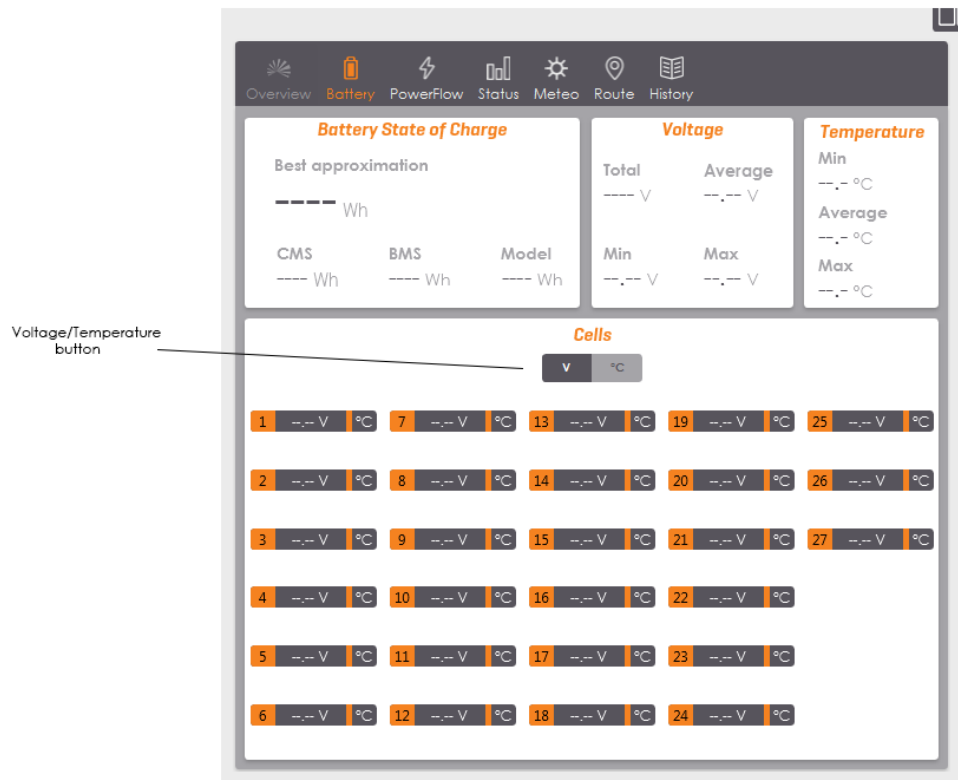


Figure 16: Battery tab in the application

The Power flow tab (Figure 17) shows the power flow between the MPPTs, BMS, CMS and motor. There are two blocks in the information pane, namely the Top Shell and Under Shell. The MPPTs, BMS and motor are made of switches which contain the value power as default. When clicked on, the view of the switch changes to current and voltage.

The battery in the Under Shell block has an image of a battery. This image will change to the value of the percentage charged accordingly.

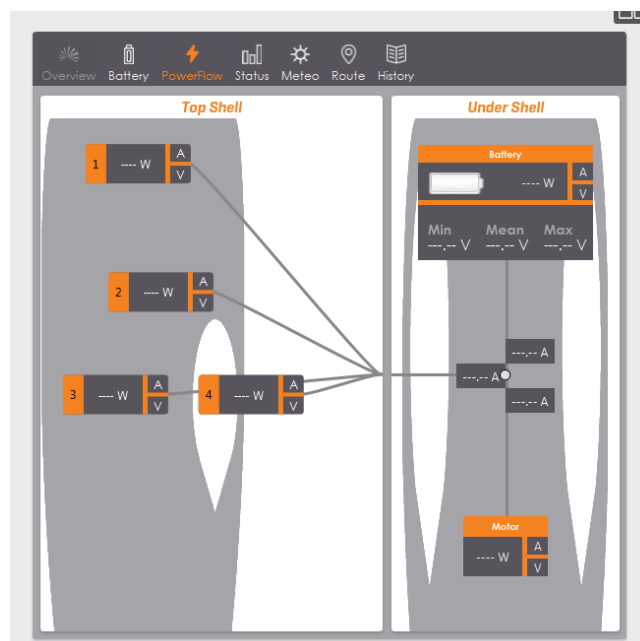


Figure 17: Powerflow tab in the application

The *Status tab* (Figure 18) shows the connection of all components between the server of the application and the Mobiboxx inside the Nuna. There are four block in the information pane, namely MPPT, Battery, Connected Sensors and Motor.

The MPPTs are again represented as switches. The default view is the status of a MPPT. When it is not connected, the switch will be red. Otherwise it will be grey and show an error flag, or when there is none, it will show 'Connected'. When clicked on a switch, the temperature will be shown.

The Battery and Motor block also show statuses. When an error flag is present, it will get privilege over a status or limit flag. The status will be red when the sensor is not connected or has an error flag, otherwise it will be grey.

The Connected Sensors block shows all sensors of the Nuna or Mission Control and the state of connection. A sensor will be red when it is not connected, else it will be grey.

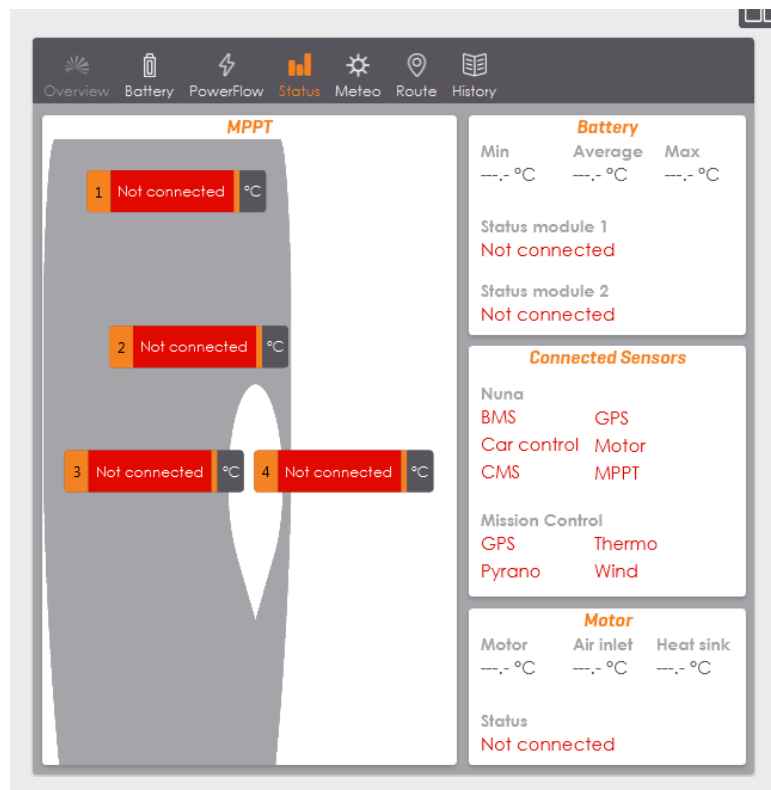
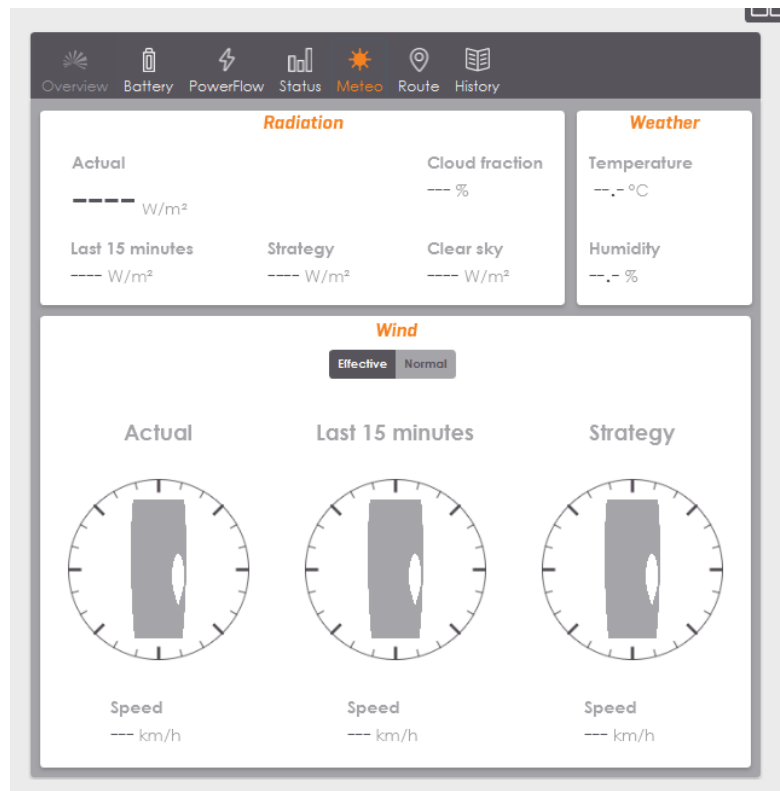
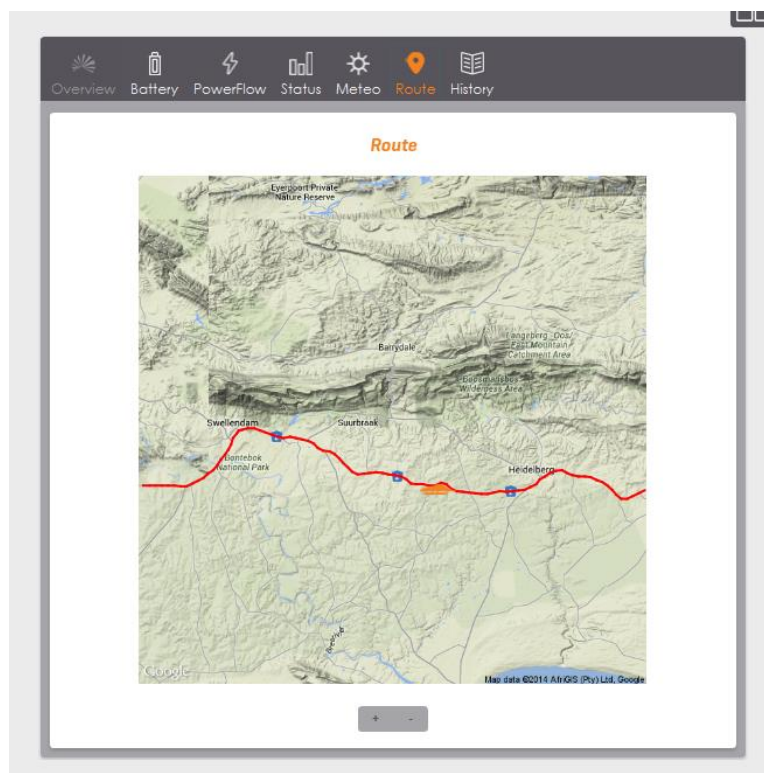


Figure 18: Status tab in the application

The *Meteo tab* (Figure 19) shows information about the weather. It consists of three blocks, namely Radiation, Weather and Wind. The Wind block contains an effective/normal button. This button is meant to choose between the views of the effective or normal wind. The default configuration shows the effective wind. When data about the wind is received there will appear arrows on each compass in the direction of the wind.



The *Route tab* (Figure 20) shows the route on a map. On the map the Nuna will be visible on the current location. The map also displays the control stops and charge stops. There are two buttons at the bottom of the information pane to zoom in or out on the map.



The *History tab* (Figure 23) has the functionality of revising saved logs of previous or current races. The button with the plus symbol will result in a popover, in which a log file can be loaded. It is then possible to load any saved log file with the extension *.mv.db* (Figure 21). When a file has been chosen it is shown as a table in the information pane. It is possible to load logs of more than one sensor (Figure 22). In the bar next to the plus button new buttons will appear, of all loaded logs, containing the name of the sensor. These buttons can be used to switch between the log files. A maximum of four log files can be shown in the history tab. When a fifth file has been chosen, the file that was loaded first will be removed from the tab.

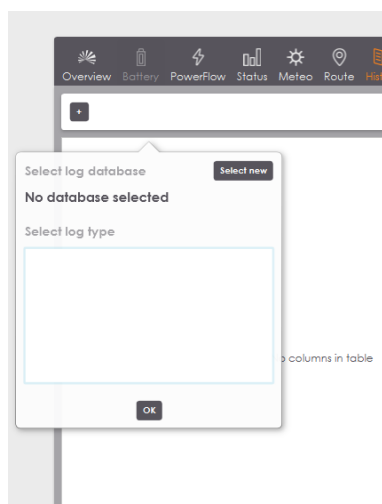


Figure 21: Popover for different log databases

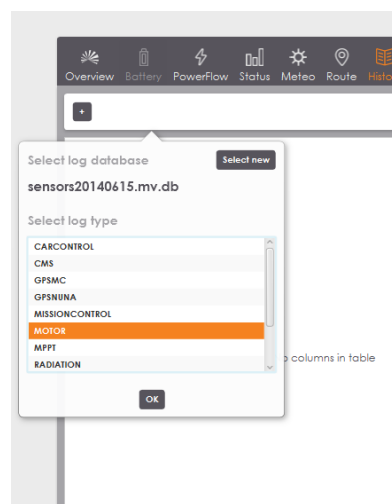


Figure 22: Logs of different sensors could be chosen

REFRESHTIME	WINDDIRECTION	WINDSPEED
2014-06-15 06:40:35	29	39,24
2014-06-15 06:40:31	33	43,52
2014-06-15 06:40:27	33	40,46
2014-06-15 06:40:23	32	51,08
2014-06-15 06:40:20	14	58,39
2014-06-15 06:40:17	1	67,14
2014-06-15 06:40:14	8	70,27
2014-06-15 06:35:54	327	26,6
2014-06-15 06:35:49	339	34,7
2014-06-15 06:35:45	330	30,06
2014-06-15 06:35:41	332	32,65
2014-06-15 06:35:37	343	33,01
2014-06-15 06:35:33	337	36,58
2014-06-15 06:35:29	338	30,17
2014-06-15 06:35:24	353	38,38
2014-06-15 06:35:20	340	30,02
2014-06-15 06:35:16	340	34,92
2014-06-15 06:35:12	329	36,68
2014-06-15 06:35:08	321	46,76
2014-06-15 06:35:04	324	41,26

Figure 23: History tab in the application

6.2.3 STRATEGY VIEW

The Strategy view makes it possible to load strategy files and show plots of different kinds of strategy data. When a strategy file has been loaded and selected for the strategy view, all lines are drawn automatically in the graphs. The strategy view contains three blocks which contains two graphs each.

At the top of the strategy view three toggle buttons can be found who each provide different functionalities (Table 4).

Tab icon	Name	Description
Battery Power In Power Out	Battery/Power In/Power Out	Change the view of the graphs to battery, power in or power out configuration(specified in config.xml)
Mid term Long term	Mid/Long term	Change the axis of the graphs on the right to mid term or long term
Time Distance	Time/Distance	Change the axis of the graphs to time or distance

Table 4: Buttons at the top of the strategy view

A graph can be enlarged or downsized when clicked on (Figure 24). When a graph is double clicked it is automatically saved as a .png file in the logs/graphs folder.

The settings button located at the middle of a block makes it possible to add extra lines to the graphs in that block. A popup will appear where lines can be chosen per value (Figure 25). The value of a graph has been set by default because of the battery/power in/power out configurations, but can of course be changed by selected the desired value in the settings popup.

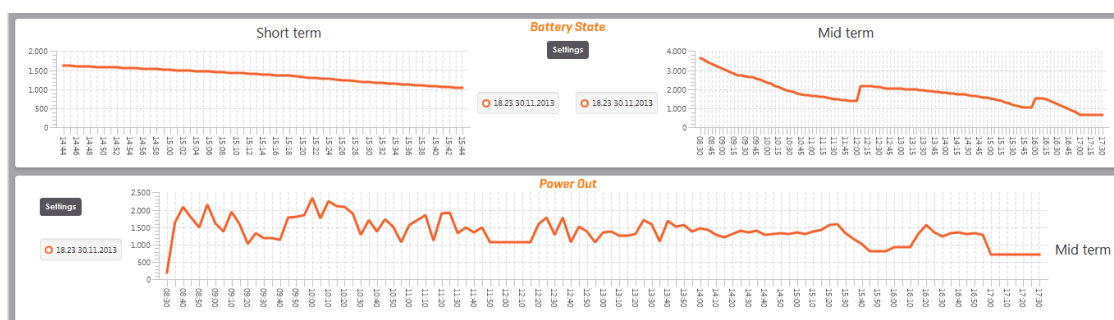


Figure 24: Enlarged graph in strategy view



Figure 25: Settings popup in the strategy view

6.2.4 TOP PANE

The top pane contains important features that always need to be visible in the application. In Table 5 below, an overview of all icons is given.

At the left size of the top pane warning icons will appear when some elements obtain critical values. In the middle the current time is displayed with the cruise control button, chat message button and the sensor/strategy button around it. On the right side the screen option buttons, the connection button and strategy file button are placed.

Icon	Name	Description
	Sensor view	Show the sensor view with two information panes
	Strategy view	Show the strategy view with one large information pane
	Cruise control	Turn the cruise control on/off and choose a speed
	Chat	Send a message to the driver of the Nuna
	Strategy file	Choose strategy files
	Connection	Turn the server on/off and see the connection state of the server and Mobiboxx
	Extra screen	Open an extra screen
	Full screen	Set the screen to full size
	Small screen	Set the screen to a smaller size

Table 5: Icons in the top pane with names and description

The extra screen button (Figure 26) makes it possible to open the application on multiple screens. Every time the add screen button is pressed, one extra screen is created.



Figure 26: Add screen button

The full screen (Figure 27) and small screen button (Figure 28) controls the size of the screen. The application starts in full screen by default. This means that the small screen button will be visible at first. When clicked, the screen size will decrease. After that, the full screen button will appear on the same spot as the small button was before. When clicked on this button, the screen will go back to full screen mode.



Figure 27: Set full screen button



Figure 28: Set small screen button

When the cruise control in the Nuna is switched on, the remote cruise control can be set in the application. To turn on the remote cruise control, click on the cruise control button (Figure 29) to open the popover (Figure 30). In the popover, click on the on / off button to turn the cruise control on. Then enter the desired speed of the remote cruise control and press Enter. When the Nuna has received the remote cruise control, the speed will become visible in the cruise control popover when the application has received the just adapted cruise control speed. To turn the remote cruise control off, press the on/off button in the cruise control popover.



Figure 29: Cruise control button

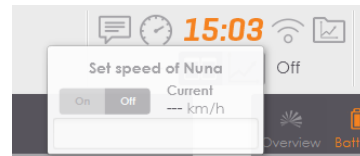


Figure 30: Popover with cruise control settings

A chat message can be sent to the driver of the Nuna. To start this process, click on the message button (Figure 31). In the popover that appears a message can be typed (Figure 32). To send the message, just press Enter.



Figure 31: Message button

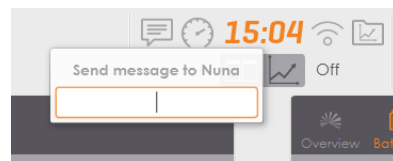


Figure 32: Popover for sending a message

The view of the application consists of the sensor and strategy view. These are already explained in 6.2.2 and 6.2.3. To change the view, click on the sensor / strategy button at the middle of the top pane (Figure 33). Now the view has changed.



Figure 33: Sensor / Strategy button

Warnings appear on the left side of the top pane when values have reached a critical region. To view which value is wrong, click on the icon that has appeared. A popover with the list of critical values will appear. When clicking on a value, the corresponding tab will open in the left information pane. The different types of warnings are listed below (Table 6).

Icon	Name	Description
	BMS warning	Warning about BMS measurements (e.g. cell voltage, cell current, cell temperature)
	MPPT warning	Warning about MPPT measurements (e.g. MPPT status, MPPT temperature)
	Motor warning	Warning about motor measurements (e.g. Motor temperature)
	Meteo warning	Warning about weather measurements (e.g. Wind speed)
	Connection warning	Warning about measures about connections (e.g. No connection)

Table 6: Icons of different types of warnings with names and description

The colors and flashing of the warning icons vary depending on their content. There are two gradations of critical values, the grey-yellow area and the yellow-red area. When a value has changed from normal to critical it is present in the grey-yellow area. When the values in the popover over a warning icon are only in this area, the icon will be yellow and non-flashing (Figure 34). A value will occur in the yellow-red area when it has become even worse. When a value is in this area for the first time, the warning icon will be red and flashing. The flashing will stop when clicked on the icon. The flashing will occur again when the value has become worse. This exact deviation of the value can be specified in the config.xml file at the *ResumeFlashing* field (section 4.2).

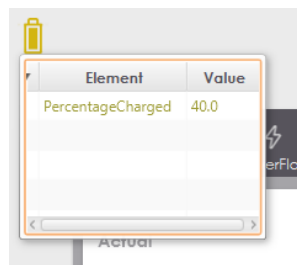


Figure 34: Popover of BMS warning

6.2.5 BOTTOM PANE

The bottom pane (Figure 35) contains a visual overview of the race. During the race the image of the Nuna will move according to the actual position. In the figure below all visible numbers are explained. The numbers are relative to the position of the Nuna and thus will change every time the position of the Nuna changes.

The distance shown beneath the loop has multiple functions. When the Nuna has not yet reached the loop it stands for the distance to the loop. When the Nuna is inside of the loop it stands for the distance until the end of the loop.

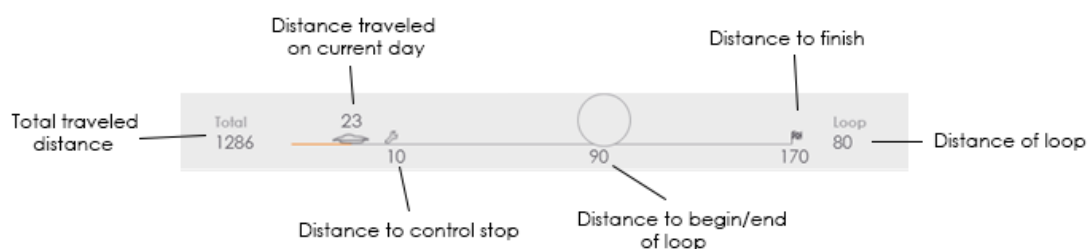


Figure 35: Bottom pane during an example race

7. CONCLUSION

7.1 PROBLEM REQUIREMENTS

A list of requirements has been made with the client (). The list of requirements defines the goal during the bachelor project. It also gives a clear overview in what was finished and what not. We have reached a great part of our goal and many requirements have been implemented. Reaching our goal shows from the following points:

- The testing days
- The weekly meetings the client

7.1.1 TESTING DAYS

The testing days were a good opportunity for a large part of acceptance testing. It gave a clear overview in which features work and which not. During the testing days bugs in the application have been found. The found bugs have been solved after the test days.

7.1.2 WEEKLY MEETINGS

The team had weekly meetings with the client discussing the progress. These meetings were of great value for both parties. The client can then see the progress the development, while the team received feedback about the development of the application. The weekly meetings were used for presenting and verifying the application. A part of the acceptance testing has been done here as well; as there are features that could be shown directly (like the strategy graphs). In respect to the client, the expectations have been fulfilled. During the meetings the team has asked the client whether we have met with his expectations, which he answered positively. This does not mean that the application is finished completely, since there are more features to implement. Because the application also connects with other hardware, changes to the application will be made. An example is that the *Nuon solar team* is getting a new all-in-one sensor which they are going to use, which will be taken in use within weeks after the delivery of the product.

7.1.3 REQUIREMENTS

There were some requirements that could not be met. This had several reasons. The first requirement to mention is a system requirement:

- Java version 7 has to be installed on the running machine

This requirement has been changed to Java 8. JavaFX has been used for the creation of the GUI. The version of JavaFX that has been used was version 2.2. However, some functions which were expected to be in the framework were not there (like setting the window of the application to full screen). The team had therefore decided to update to the most recent version, which was JavaFX 8. In order to use JavaFX 8, Java 8 has to be installed because JavaFX 8 is a built-in library in Java 8. This turned out to be a good choice because it made the development of the application a little easier, but also more robust as JavaFX 8 has less bugs than its predecessors.

The following feature is a feature in the requirements list (user requirements) that has not been implemented:

- A setup for automatically saving preferences of the car, weather and country

The reason for not implementing this feature is because of time constraints. The amount of time during the bachelor project did not allow us to implement this feature. There are however other features that are not implemented as well, but the reasoning behind it is different. Other features that have not been implemented are:

- Maximum shortwave radiation
- Minimum shortwave radiation
- Visual representation of fraction deviation
- Visual representation of the bottom shell in the status view

During meetings with the client he came to the conclusion that some features were not needed anymore. The ones not needed are the first three items in the list mentioned above. The last one in the list was an idea which the team came up with. Implementing the status pane slightly different would give a much broader overview of the status of different car components. After asking the client, he agreed with this extra feature. Furthermore, there are the following features which have not been implemented:

- Auto zoom functionality in the route view
- Auto zoom functionality in the strategy view

During one of the meetings with the client the team asked him whether he would still be satisfied if the application had no auto zoom functionality, as the manual zoom was another feature. He agreed, and accepted moving the auto zoom functionality to a later version of the application. Then there are all the features regarding the weather and satellite:

- Display of generated weather maps
- Display of layers in the basic map
- Automated detection of available weather models and available runtimes
- Display of satellite images (infrared, radiation prediction)
- Animated weather charts
- Multiple formats for the application
- MT-SAT prediction of radiation
- Satellite panel

These features are also not implemented. Reason for this is because the former application designer, together with the *Nuon solar team*, decided to develop a standalone application for the satellite and weather functions. It was therefore not needed to implement these features, as the client said. We did however have a good idea about the still to be made satellite application. If the application would be finished on time, we would see if there was a possibility to integrate this application in our program. This would have been the best and client also liked the idea. Unfortunately, it did not happen because the satellite application is still not finished and could therefore not be integrated in our application.

Most of the requirements are implemented, satisfying both the client and the team. Would we have had more time for development, then we could have definitely implemented all features and even some extra ones, as the client said that the application can never be finished.

7.2 PRODUCT VALIDATION

The results of the project needed to be validated by the client as well as the programming team. Firstly the application was tested thoroughly during testing days. During the testing days a test plan was created () which formed a list with many points that needed to be checked. Many of these points were directly copied from the list of requirements as a validation test. At the end of the project not all of these points were implemented and therefore not all tested to be good. However, the client has reviewed the list of requirements and accepted it. This reviewing happened in a meeting together with the programming team. The meeting formed our acceptance test.

The deliverables, for the project as well as for the client, were completed and handed in. The deliverables for the client were directly used, because the new Nuon Solar Team, Nuna 8, has one full time member who is going to extend the application. There is also a meeting planned to inform the Nuon Solar Team about the manuals and to make agreements about further cooperation.

7.3 LEARNINGS

The project was an amazing learning experience and our background in Computer Science especially came in handy. We have learned a lot during our bachelor project. It is therefore a good idea to divide all the learnings into the following subparts.

- Learnings during the research phase
- Learnings in the development phase
- Learnings in the concluding phase

7.3.1 LEARNINGS FROM THE RESEARCH PHASE

The main goal of the research phase was to gain a clear overview in how the development was going to be processed. The research phase gave us a great opportunity in learning from these different areas:

- Programming language
- GUI frameworks
- Network connection frameworks

Different programming languages have been researched in the research phase. LabVIEW is one of these languages. LabVIEW is a graphical programming environment, with one of the advantages that

its source code is very similar to circuit diagrams, thus making it an easy language for electrical engineers to pick up. Although the language supports Windows 7 and 8 and does not require the user to install various extra programs, it does require future software development teams to have access to LabVIEW. This is not just a language, it is a programming environment and the LabVIEW language could thus only be accessed in the corresponding IDE. Thus future access to LabVIEW is not completely assured which hugely decreases the extendibility of applications in this language.

Another language that was researched was C#, which is similar to Java. The disadvantage to C# is its small library (unlike Java). We now have a general idea of these programming languages, which in the end were not used.

We have also learned a small amount about GUI frameworks. One of these frameworks is QT Jambi, which supports Java. This framework was however not suitable because creating basic elements took a lot of time. Then there was Swing which is a Java build-in framework for creating GUIs. The problem with Swing is its difficulty to use. Finally there was JavaFX, the GUI framework used for the project. This framework is well documented and makes it easier to create a GUI. But especially its application to create a GUI, the JavaFX Scene Builder, comes in handy. The reason for this is that the Scene Builder is a What-You-See-Is-What-You-Get application to create GUIs without codes.

The network connection frameworks were also a part of the research phase in which we have learned. There were two network connection frameworks, one extending the other. There is Java.net which offers functionality for sockets, and there was javax.net, an extension of the first one, which added functionality for the ServerSocket and a more secure connection.

7.3.2 LEARNINGS FROM THE DEVELOPMENT PHASE

The development phase was the phase in which was learned the most. The design challenges in section 4.5 gave us a good opportunity to learn from developing the application. We have certainly learned from these challenges. The learnings we have made out of the challenges are:

- *Challenge 1:* We now know how to connect with serial sensors. The experience we had so far was mostly with developing applications for computer use only. It is a good learning experience to have made a real life product.
- *Challenge 2:* we also learned a great deal in how data is stored and retrieved efficiently.
- *Challenge 3:* Developing a GUI with this extent was not done earlier. We had some experience in using Swing, a build-in GUI library in Java, but now, because of this project, also with JavaFX. JavaFX is a great framework for building graphical user interfaces with Java but we did not use it before. Now, it is our preferred way for designing a GUI with Java. This will be of good use for future java projects.
- *Challenge 4:* An important this for us is that we wanted to see whether we could apply any of the bachelor background to this project. So far this has been worked out positively, as can be seen in the design challenges. We can now see that putting effort in your education pays off greatly.

7.3.3. LEARNINGS FROM THE CONCLUDING PHASE

Some learning was also made during the final phase of the bachelor project. The learnings apply in the following areas:

- Writing a clear test plan
- Writing manuals
- Requirement reviewing with the client
- Validation testing

Writing our test plan

Most of the testing has been done in the concluding phase. Because the end of the project was coming closer, we had to make sure that the application has been tested thoroughly. Writing a test plan was therefore necessary (). Using a test plan is a very good practice, as it gives a clear overview in what still needs testing, and what the deadline is.

Writing manuals

For the user the application is a black box with some text and buttons. The user needs to have a good understanding of what the application can do. For this purpose, a *User Manual* needed to be made, something that we have not done earlier. Making the *User Manual* was tougher than initially expected, because the sections in the manual had to be formulated in a way that someone without any programming language would understand. In our entire Bachelor, we have not done any documentation that is so purely intended for users.

It is not only the user, but also the technical staff and future developers. For all these purposes, manuals have been made. The manuals consist of the above mentioned *User Manual*, an *Extensibility Manual* and a *Technical Manual*. Writing manuals is very important as we will likely be less involved in future developing.

Requirement reviewing with the client

Another point that we have done in the concluding phase was reviewing the list of requirements with the client. This is done by going through this list and checking off as many points as possible. This is a very important point, because the client will get a good idea to which extent the requirements have been realised.

Validation testing

Learning has also been done with validation testing. You can see the validation tests as real time tests. With that we mean testing the actual application with the Nuna solar car and see what happens. Validation testing gave us the opportunity to see the product in action and to make it more robust.

7.4 USED KNOWLEDGE

The Object Oriented Programming course and the course of Algorithms and Data structures gave us a clear start for developing, especially because Java was used in those courses and also in developing the application.

Furthermore, there is a particular course in our bachelor that we took great advantage of: Software Quality and Testing. This course has been helpful for us in several ways:

- Using maven
- Using mocks to fake objects
- Unit testing
- Integration testing
- Using git

8. DISCUSSION AND RECOMMENDATIONS

This section contains the discussion of the project. First, our product is critically discussed (section 8.1) as there are certain items of the product that could have been done better. Then, the process is reflected (section 8.2). Finally, several recommendations are given to future developers (section 8.3).

8.1 PRODUCT DISCUSSION

One of the aspects of the application that does not work as planned is the stability of the connection of the application. The connection works in a normal situation, but there are some corner cases that can break the connection between either the server and the Mobiboxx or the connection between the server and the clients. The team tried to make the connection stable by testing the connection thoroughly, but since the team members have never before implement a client-server connection model, the current implementation may not be optimal.

An aspect of the application that became unexpectedly a problem is the start-up time of the application. The amount of FXML files that are loaded during the start-up of the application increased during the project, which takes some time to load. The average start-up time of the application is roughly 25 seconds. It has been tried to decrease the start-up time by trying to store preloaded FXML files. This would reduce the start-up time significantly. However this turned out to be impossible to do. A loader screen has been designed to show the user the application is starting up.

When an exception occurs in the application, some explanation is printed in the command line of the application. However, when the user starts the application by directly running the jar file, no command line is available for printing the error messages. The user has to run the jar file from the command line to receive error feedback. The team preferred to have the exception messages displayed in the GUI of the application, but had insufficient time to implement this feature.

8.2 PROCESS REFLECTION

An aspect of the process that could be improved is the planning of the team. There had been planned too little time for both the start of the development phase and the concluding phase. The items planned for the first week took the team two weeks to complete. The team also did not plan enough time for the concluding phase. The last weeks of the process team members had to work during weekends and evenings to complete the items in time.

Another part of the process that should have been done better is the communication with the Nuon Solar Team. There was some difference between the expectations of the team and what the Nuon Solar Team offered the team. For example the availability of the solar car and the amount of testing days was less than what the team expected. This was however due to lack of communication between the team and the Nuon Solar Team.

A third point that would have improved the process is simulating the production environment in a very detailed way. The team captured input data from the Mobiboxx by saving it in a text file. It would have been better if this was done earlier and if the file was read at the same time interval as the data was captured.

8.3 RECOMMENDATIONS

There are still a lot of features for the application that have not (yet) been implemented. Future development teams can continue with these features. The following features have not been implemented:

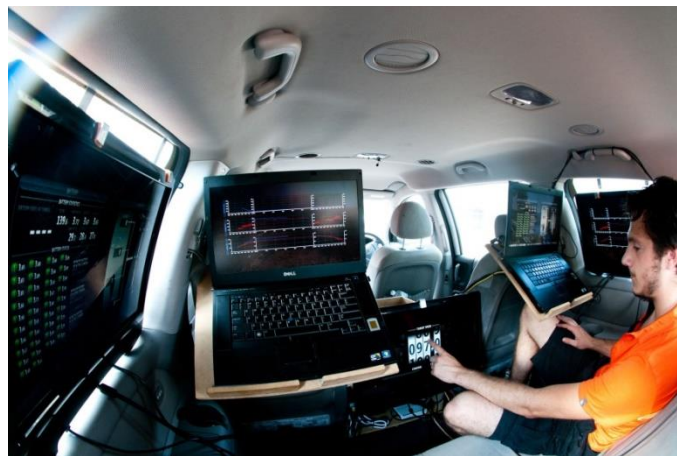
- Brighten the MPPTs of the *Powerflow* tab based on the power in per MPPT
- Add animation to the GUI
- Integrate real time strategy calculation in *Matlab* in the application
- Stream race data to an smartphone app for the people watching the race
- Setup where the user can change the *config.xml* file settings in a more user friendly interface

9. REFERENCES

- [1] <http://www.eclipse.org/>, retrieved 22 June, 2014.
- [2] <http://www.eweek.com/c/a/Application-Development/Microsoft-Ends-Support-for-Visual-Basic-60/>, retrieved 22 June, 2014.
- [3] <http://www.mathworks.nl/products/matlab/>, retrieved 22 June, 2014.
- [4] <http://maven.apache.org/what-is-maven.html>, retrieved 22 June, 2014.
- [5] <http://junit.org/>, retrieved 22 June, 2014.
- [6] <http://www.eclEmma.org/>, retrieved 22 June, 2014.
- [7] <http://git-scm.com/>, retrieved 22 June, 2014.
- [8] <https://github.com/>, retrieved 22 June, 2014.
- [9] <http://www.h2database.com/html/main.html>, retrieved 22 June, 2014.
- [10] <http://stackoverflow.com/questions/24010737/why-are-added-tables-from-java-not-visible-when-i-open-the-h2-console>, retrieved 30 May, 2014.
- [11] <http://visualvm.java.net/>, retrieved 22 June, 2014.
- [12] <http://www.wireshark.org/>, retrieved 22 June, 2014.
- [13] <http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html>, retrieved 22 June, 2014.
- [14] http://www.adobe.com/nl/products/photoshop.html?kw=semgeneric&sdid=JTZDE&skwid=AL!3085!3!48065426064!e!!gl!adobe%20photoshop&ef_id=wtRPQplCoSgAAMBv:20140627195724:s, retrieved 22 June, 2014.
- [15] <http://office.microsoft.com/nl-nl/visio/>, retrieved 23 June, 2014.
- [16] <http://astah.net/>, retrieved 23 June, 2014.
- [17] <http://www.visualobject.com/>, retrieved 23 June, 2014.
- [18] http://en.wikipedia.org/wiki/Dropbox_%28service%29, retrieved 24 June, 2014.
- [19] <http://en.wikipedia.org/wiki/Trello>, retrieved 24 June, 2014.
- [20] <http://programmers.stackexchange.com/questions/190482/why-use-a-database-instead-of-just-saving-your-data-to-disk>, retrieved 24 May, 2014.
- [21] <http://en.wikipedia.org/wiki/ACID>, retrieved 28 May, 2014.

BUILDING A GUI FOR NUNA'S MISSION CONTROL

PLAN OF ACTION



CLIENT

Rune van der Meijden

CONTACT

rune@nuonsolarteam.com

SIGNATURE

FORMER APPLICATION DESIGNER

Noud Brasjen

noud.nuna6@gmail.com

SOFTWARE DEVELOPMENT TEAM

Isha van Baar

I.J.vanBaar@student.tudelft.nl

Zakaria el Bakkali

Z.elBakkali@student.tudelft.nl

Jurian van Dalfsen

J.vanDalfsen@student.tudelft.nl

Eveline van der Schrier

E.D.vanderSchrier@student.tudelft.nl

Lodewijck Vogelzang

L.G.J.Vogelzang@student.tudelft.nl

TEAM COACH

Nicolas Dintzner

N.J.R.Dintzner@tudelft.nl

PREFACE

On February the 17th, the contents of the Computer Science Bachelor Project were revealed. This project, also referred to as "Bachelor Eindproject" or "BEP", is the course that finishes the Computer Science curriculum at Delft University of Technology. It is carried out by groups which are given the assignment to work together like a software development team, producing a real product to be used by a real client, solving a real-world problem.

One of the available projects focussed on the Nuna Solar Team and their solar car, the Nuna Solar Car. This solar car is very famous at Delft University of Technology and it immediately gained the interests of our project team.

The purpose of this document is to provide both a contract between our project team members and the Nuna Solar Team client, but also a contract between the team members and their team coach. It is comprised of formal information about our working methods, organisation and communication. Different parties will have to agree with this document and provide a signature of approval.

Firstly in the introduction the context and motivation of the project team is explained. Then, the required deliverables are formed. After that, the stakeholders and the project itself are respectively described. Last of all, it is attempted to give an assurance of the products quality. In the appendix, a more thorough description of our the requirements of our product is given.

INTRODUCTION

CONTEXT

The Nuon Solar Team is a student team set up to show the world what is possible in the area of high-end technology and green energy. Therefore, every two years the Solar Team builds an improved solar car, the Nuna Solar Car. With this solar car the team competes in the World Solar Challenge in Australia. Starting in 2001, the team won four times in a row. Their latest victory was achieved in 2013 with the Nuna 7 Solar Car.

Part of the success can be traced back to the concept of a chase car called 'Mission Control', which follows the Nuna. In the chase car, a laptop application receives all kind of telemetry and weather information from the Nuna car and several other sensors. This way the students can give the driver advice and anticipate their strategy.

This year, the team will compete in the Sasol Solar challenge in South Africa for the first time. This is done as an experiment to improve problems of the Nuna 7 in order to build a perfect Nuna 8. However, the application used in the Nuna 7 chase car also showed some problems. This is where this project begins.

MOTIVATION

The team has to re-engineer a graphical user interface (GUI) for the chase car that follows the Nuna Solar Car. According to the Nuna team, the current implementation of the GUI is difficult to extend and maintain, giving difficulties at any change like different car parts, different car layouts and different environments.

The Nuna team has given several reasons for these problems themselves. The main problem of the current implementation of the GUI seems to be the Visual Basic 6.0 language, which is not supported anymore. Moreover, the application is written by one non-programmer, the former application designer, who did not provide any documentation or code comments and who hardcoded a great part of the user interface.

The most important task expected by the Nuna team is thus not to come up with a completely new program and additional features but to come up with a re-engineered version of the current GUI, coded so that it is easily extendable and maintainable for any Nuna Solar Car changes.

APPROVAL

The approval of this action plan is done by asking each of the team members to read this document and, upon agreeing, place a signature.

DELIVERABLES

Our deliverables consist of both a product and documentation.

PRODUCT

The expected delivery is a fully functional application for the chase car that follows the Nuna Solar Car. The GUI of this application shows telemetry and strategy information gained from data received by various sensors on both the Nuna Solar car and the chase car. The application is to be used by the Nuna Team and should not be in any way readable or controllable to other solar racing teams.

The current implementation of the GUI for the Nuna chase car works well and thus its front end should not be changed a lot (*Appendix: List of Requirements*). The application rather has to be re-engineered to make it easier to improve later on by others. It should be extendable by another software requirement team in such a way that the addition of functionality would not harm any existing features. Therefore the application will be rebuild from scratch.

The Java program language will be used because it is a known program language with lots of frameworks and support. This makes it unlikely that Java will soon disappear. The entire team is also familiar with Java and able to design maintainable and extendable Java code. Other languages and programming environments are taken into consideration as well. A thorough description of the team's language and framework choice can be found in the *Research Phase Report*.

To make the code understandable the team will try to create a clear well-thought structure based on our initial models like class diagrams and component diagrams. All models and further documentation will be provided in the product package to help future software development teams change, extend or maintain the application. Further requirements of the application and its users can be found in *Appendix: List of Requirements*.

DOCUMENTATION

Several reports and models will be made to provide the client, team coach and future software development teams with sufficient code information. These reports include this report, the *Research Phase Report*, the *List of Requirements* (*Appendix: List of Requirements*) and the finishing *Bachelor Project Report*. Additionally, the team's *Final Presentation* will give every stakeholder and an additional Bachelor Project course leader the opportunity to see the team's final product and review its quality.

The *Research Phase Report* will be the constructed report at the end of the research phase. It will explain our research and the corresponding results. The research phase report will be mostly used by the software team itself as the outcomes will be needed during the development of the product.

The *List of Requirements* (*Appendix: List of Requirements*) will provide a detailed overview of the requirements that the final product must, could, should and would have. It will be used by the software team during the product development but it will also be of great importance to the client to validate the product.

The *Bachelor Project Report* will consist of the findings during the project, what the challenges and opportunities were and the solutions the software development team came up with. As the team will have to send its code to SIG for evaluation, the *Bachelor Project Report* will also include the feedback and the team's reaction. The report will be meant for the team coach in particular. It will be composed during the Bachelor Project and finished at the end.

Internally in the software engineering team, a *Journal of Tasks* document with corresponding schedule will be used to keep track of the work per week.

STAKEHOLDERS

In order to realise the delivering of the products, all the people that are associated with the project and their given requirements are listed.

SOFTWARE DEVELOPMENT TEAM

The software development team is in charge of the specification, implementation, testing and documentation of the specified product. The members of the development team are expected to work full-time on this task, attend meetings and keep promises.

Zakaria, one of the team members, performs a slightly different role. Since he still has a lot of other courses to finish and could therefore still be rejected to the Bachelor Project, the software development team expects that Zakaria will regain any lost time that he spent on these courses.

Furthermore, any of his work needs to be documented closely so that the other members will not get any problems in understanding it.

CLIENT

The client is expected to give a clear overview about what has to be done and what he wants. As the client is one of the few people that worked with previous versions of the code, he is expected to clarify any question that the team has. The client is also expected to attend the weekly meetings with the software development team.

FORMER APPLICATION DESIGNER

This person plays a key role in explaining the previous version of the product. He has to be available in the case that we do not understand the current code and want to ask something about it. His role is important because there are no comments in the code and there is also no official documentation whatsoever. The team does not expect him to be available on a fulltime basis but his availability should be reasonable. He should also be willing to have certain meetings with the team (on request and with approval on both sides).

TEAM COACH

The team coach is expected to direct the team through the process of building and delivering the products. If the team coach sees that the process is not going well he should communicate this to the team. He is also expected to be a source of information in regard to software engineering and also has to attend weekly meetings with the team (and in some cases also with the client).

PROJECT

DEVELOPMENT METHOD

In order to fulfil the product requirements, the software development team will work on the project for thirteen weeks. During this project the Scrum method is used. This agile management method is performed by making a product backlog list of all product features. The team uses sprint iterations of one week to decide the features that have to be implemented that week. Every week, a prototype is delivered with new features.

PLANNING

To make sure the product is finished in time, a general planning has been made. This planning will merely serve as a help for deciding the importance of each product feature and not to be always followed. The first three weeks will be the research phase. After a short break to learn for exams, the next eight weeks will be the development phase. The last two weeks will be used to test the application. A schematic overview is given in *Figure 1*.

	Research phase			Exams	Development phase										Test phase		Extension
Product Backlog / Sprint week	11	12	13	14-16	17	18	19	20	21	22	23	24	25	26	27	28-29	
Documenting List of Requirements																	
Documenting Plan of Action Report																	
Reverse engineering of current application																	
Researching Java GUI frameworks																	
Researching Java real-time connections																	
Documenting Research Phase report																	
Modelling the new application																	
Creating a framework																	
Connecting application to mobiboxx																	
Sending, requesting mobiboxx data																	
Showing data in some way in framework																	
Creating overview panel																	
Creating battery panel																	
Creating power flow panel																	
Testing application with driving solar car																	
Creating strategy panel																	
Creating remaining panels																	
Sending code to SIG for evaluation																	
Creating remaining features																	
Implementing SIG code feedback																	
Composing Bachelor Project Report																	
Testing thoroughly																	
Booking presentation room and inform																	
Resending code to SIG																	
Presenting the final product																	

Figure 1: Schematic overview of likely planning

The research phase consists of three weeks where the software development team prepares itself for the development of the product. In week 11 the team will work on the following tasks:

- Writing the Plan of Action document
- Creating a requirements list with user and system requirements (see Appendix: List of Requirements)
- Having meeting sessions with the client and others from the Nuna team
- Being able to run the current source code on our own computers
- Reverse engineering the architecture of the current application implementation

In week 12 the team will work on the following tasks:

- Researching GUI frameworks in Java
- Real-time connection research in Java
- Reverse engineering the architecture of the current application implementation

In week 13 the team will work on the following tasks:

- Modelling the new application
- Documenting all of our findings in the Research Phase report
- If necessary more research that has not been completed in the second week

The development phase consists of eight weeks where the software team develops the application. In week 17 the team will create a framework for the application. The application will be able to make a connection to the Mobiboxx and send and request data. In week 18 until week 20 the team will create the most important panels: the overview panel, battery panel and power flow panel. At the end of week 20 the team will test the application with the solar car. In week 21 the strategy panel will be implemented. From week 22 until week 24 the team will create the remaining panels and send the code to SIG for evaluation. Finally in week 25 any remaining features and code feedback will be implemented. If during the development phase it would occur that there are not enough tasks for every member of the software development team, a part of the team will work in advance on tasks of the next week. This might lead to the delay of tasks, since not the whole team will be working on them. The software development team will closely monitor the planning and make sure it will not be delayed.

The test phase consists of two weeks where the team will thoroughly test the application and compose the Bachelor Project Report based on earlier written documentation counterparts of product backlog tasks. For this the team will need a driving Nuna 7 Solar Car with chase car. If any bugs are found during the tests, there is a possibility to solve them. Finally after this phase, the final product is delivered.

Nonetheless, there might be a possibility that the team wants to finish certain features or add new ones. Therefore it could be possible that the first weeks of the summer holiday are also spent on this project as a possible project extension.

PLANNING CONSTRAINTS AND FREEDOM

All the team members will be working daily, five times a week for ten weeks. Each day consists of 8 hours, which gives a total of 400 hours. Additionally there will be the additional research phase, performed in quarter three to give the team a head start, consisting of three weeks of research equalling a total number of 60 hours.

Each of the team members will have a certain freedom in not showing up. If the situation occurs wherein a team member cannot possibly show up, it should be clearly notified at least one week prior to the absence. There will be no formal restrictions in how many times a team member is absent but the amount of work should be the same for every member of the software development team.

There will also be rules that are made between the software team members. When a member arrives after the starting time, usually 09:00, the team will not accept this delay and confront the team member with this behaviour. Apart from this warning, this member is then obliged to bring delicacies such as candy, pie and ice cream. Also, every Friday someone of the team must bring a treat as well.

MEETINGS

During the project the team assumes to have access to the client and other members from the Nuna team. There will be a weekly meeting with the client, where the current prototype is presented, and the team coach. These meetings will be held separately although the team coach will always have the possibility to join the client meeting. During the meetings, the team will have the possibility to ask questions.

There will also be internal team meetings twice a day. The first meeting of fifteen minutes will be held in the morning when the team comes together, and the second meeting of five minutes will be held at the end of the day. These meetings will be mostly informal.

All meetings will have a chairman and a minutes secretary. The chairman will be responsible for the meeting's preparation and progress. The minutes secretary will be responsible for updating the internal *Journal of Tasks* document, its schematic counterpart and the insertion of the task division in Planbox. The important (formal) meetings will be with the team coach and client.

UNEXPECTED SITUATIONS

There is a possibility that a situation occurs where a team member could fall out due to some reason. If such a thing happens damage could be done and other team members would have more work than expected. The damage is restricted by documenting and discussing the tasks that each member has, so that the extra amount of work could be done more quickly.

In the situation that a task takes more time than planned, it is possible that other tasks are delayed too because they depend on the delayed task. To prevent this situation, the software development team will implement a framework for the application during the first week of the development phase. All panels will only depend on this framework. By finishing the framework, all other tasks will not be delayed because of dependencies on other tasks.

RESOURCES

The team will have access to the most recent version of the Nuna Solar Car, Nuna 7, and all its components so that the application can be tested regularly. All extra sensor devices like the thermometer and the pyranometer will be present.

Other application specific resources are five laptops, a private repository for code (git) and documentation (Dropbox), a task-division service (Planbox) and application environment programs (Visual Basic 6.0, Eclipse, Jenkins, Maven, etcetera).

In addition the team will also experience at least one full testing day where it can test the application during a test drive of the Nuna 7 Solar Car.

If truly necessary resources have a certain cost that could not be avoided, then the foundation will cover these costs. Thus any costs in regard to resources and travelling to test locations will be financed by the Nuon Solar foundation.

OWNERSHIP OF CONTENTS

All product deliverables are owned by the Nuon Solar team. However, the team will get all credits and informal recognition.

Sensitive information of the Nuon Solar team is property of the Nuon Solar team, i.e. codes to get access to the Mobiboxx. All official documents written by the team will not be published or will be censored by the Nuon Solar team before publishing. All documents are fully available for the team, the team coach and the Bachelor Project course leader.

ASSURANCE OF QUALITY

To assure the quality of our product, every member of this project, including the client and team coach needs to contribute his part. Now some informal requirements for each of these members who are actively involved are provided.

First of all, the team will assure the quality of the product by using our knowledge about maintainable and extendable robust programming with common standards. There are two frequently made programming mistakes highlighted by SIG which the team will try to avoid in particular. First of all, creating longer methods that provide functionality that could be refactored to separate methods. Secondly, creating a structure that does not separate production code and test code, or a directory level that does not separate the several functionalities. Furthermore the team will always try to communicate clearly to avoid miscommunication. Lastly the planning of our weekly Scrum meeting will be attempted to follow and any problems will be made clear directly to the client and/or team coach.

The client will be present at meetings and listen to our progress and problems. He will help us in any way, in both knowledge and resources, including contact with the former application designer.

The former application designer will try to answer our questions about the current application as much as possible. In this way he is obliged to communicate in any possible way (e-mail, telephone calls, meetings, etcetera) to try to solve important issues.

The team coach has a slightly different role. To assure the product's quality, he will give the team advice in regard to software engineering. Moreover, if the team shows any difficulties at using knowledge from unknown software areas (embedded systems, network programming), the team coach will try to provide information, either by advising the team himself or by arranging a specialist in the corresponding software area. Finally the team coach keeps track of our progress in regard to the official planning of the Bachelor Project.

Lastly to avoid any risks and arguments the following resolutions are proposed:

- Daily meetings are organised to ensure that every team member informs the other members about his/her progress and knows about the progress of the other members
- Weekly meetings are organised to ensure the team's progress and discuss any problems. In this way, the team coach and client will stay up to date
- When the team needs external help this will be made clear to both the team coach and client and they will try to provide this help
- Any possible arguments are immediately communicated to all team members and tried to solve internally. If this does not succeed, the team will communicate the issue to the team coach who helps to solve it

APPENDIX: LIST OF REQUIREMENTS

USER REQUIREMENTS

The users of this application are members of the Nuna Solar Car strategy team. The user has a thorough understanding of the strategy and the data displayed in the application. Therefore the application does not contain explanations of the displayed data. The user does not necessarily have an understanding of how the application works. Therefore the application runs flawlessly.

SYSTEM REQUIREMENTS

- Application runs on Windows version 7 or 8
- Java version 7 has to be installed on the machine where the application is running

- The machine on which the application is running needs to have the possibility to use a Wi-Fi connection
- The application could be easily controlled on touch screens
- When the application is not connected to the Mobiboxx, but it is connected to the sensors of the chase vehicle, the application will show estimates for the current power in and power out
- Application uses one connection to the Mobiboxx. One server application is connected to the Mobiboxx. Multiple client applications can be connected to the server application
- The application is available offline
- The properties of the solar car can be adapted in a config file. These properties are:
 - Car layout (images)
 - Battery package (layout, number of modules, number of cells, number of temperature sensors, battery state of charge-temperature curve)
 - MTTPs (number of MTTPs, position)
 - Model of power in
 - Model of power out
 - Mass of the car
 - Route (coordinates and locations of control stops)
 - Possibly other factors

FEATURES

To create a list of features we use the MoSCoW method. This means we divide all features in the following three categories: must have, should have and could have. In the 'must have' category you can find the features that definitely have to be in the final product. The 'should have' category contains the features that have to be in the final product, but not critical. The 'could have' category contains the features that are nice to have, but not necessary. The 'would have' category contains features that are unlikely to be implemented but that might come in handy for the next team.

MUST HAVE

This paragraph contains the features available in the GUI. The features are divided into the different components of the GUI.

General

- The following features are always visible: travelled distance, distance to finish, distance to the next control stop, current time
- All data is logged

Overview

Speed:

- Actual speed (km/h). When clicked: speed of motor, GPS of Nuna and GPS of MC (Mission Control, the chase car)
- Target speed (km/h). Can be changed on click
- Remote cruise control (on/off)
- Speed limit (km/h)
- Average speed over the last fifteen minutes (km/h). When clicked: speed of motor, GPS of Nuna and GPS of MC
- Expected speed in the strategy (km/h)

Battery state:

- Actual battery state (Wh). When clicked: CMS, BMS, model
- Battery power (W) . When clicked: CMS, BMS, model
- Expected battery state in the strategy (Wh)

Power in:

- Actual power in (W) . When clicked: CMS, BMS, model
- Average power in over the last fifteen minutes (W). When clicked: CMS, BMS, model
- Expected power in in the strategy(W)

Power out:

- Actual power out (W) . When clicked: CMS, BMS, model
- Average power out over the last fifteen minutes (W). When clicked: CMS, BMS, model
- Expected power out in the strategy(W)

Connection state:

- State of connection with the Mobiboxx
- Message bar

Meteo

Shortwave radiation:

- Actual radiation (W/m²)
- Cloud fraction (%)
- Amount of radiation with clear sky (W/m²)
- Maximum (W/m² and %)
- Minimum (W/m² and %)
- Visual representation (bar) of fraction deviation
- Average radiation over the last fifteen minutes (W/m²)
- Expected radiation in the strategy (W/m²)

Effective wind:

- Visual representation (compass) of the direction of the solar car and the direction and speed of the effective wind
- Actual speed (km/h)
- Average speed over the last fifteen minutes (km/h). When hovered over: visual representation in the compass
- Expected speed in the strategy (km/h). When hovered over: visual representation in the compass

Wind:

- Visual representation (compass) of the direction of the solar car and the direction and speed of the wind
- Actual speed (km/h)
- Average speed over the last fifteen minutes (km/h). When hovered over: visual representation in the compass
- Expected speed in the strategy (km/h). When hovered over: visual representation in the compass

Temperature:

- Temperature (°C)
- Humidity (%)

Battery

Battery statistics:

- Battery state of charge (V). When clicked: CMS, BMS and model
- Largest amount possible (V)
- Maximum amount (V)
- Average amount (V)
- Minimum amount (V)
- Maximum temperature (°C)
- Average temperature (°C)
- Minimum temperature (°C)

Battery status:

- Amount per battery package (V). When clicked: temperature (°C) in the appropriate colour
- Per module:
 - Module status (error message/warning) in the appropriate colour
 - Current (A)

Power flow

- Visual representation of the solar car (top view), both the outside and inside
- Visual representation of the power flow between the MPPTs, battery and the motor
- Amount per MPPT (W). When clicked: voltage (V) and current (A)
- Battery:
 - Visual representation of the amount left in the battery
 - Amount (W). When clicked: voltage (V) and current (A)
 - Maximum voltage (V)
 - Average voltage (V)
 - Minimum voltage (V)
- Current per CMS (A) of the MPPTs, battery and motor
- Amount in the motor (W). When clicked: voltage (V) and current (A)

Status/temperature

- Visual representation of the solar car (top view), both the outside and inside
- Status per MPPT (error message/warning) in the appropriate colour
- Battery:
 - Maximum temperature (°C) in the appropriate colour
 - Average temperature (°C) in the appropriate colour
 - Minimum temperature (°C) in the appropriate colour
 - Status (error message/warning) in the appropriate colour
- Motor:

- Temperature of the motor(°C) in the appropriate colour
- Temperature of the air inlet (°C) in the appropriate colour
- Temperature of the heat sink (°C) in the appropriate colour
- Status (error message/warning) in the appropriate colour

Route

- Visual representation (map)
- Current position
- Route including control stops
- Zoom in/zoom out/auto zoom

Strategy

- Diagrams with the following data visualized as a function of time or distance:
 - Battery state of charge: CMS, BMS, model
 - Power in: CMS, MPPT, model (pyranometer)
 - Power out: CMS, motor, model (windmeter)
 - Speed: motor, GPS Nuna, GPS MC, target speed, speed limit
 - Effective wind speed: windmeter
 - Cloud fraction: pyranometer
- Visual comparison with the predicted strategy values
- Diagrams are separated in two parts: one zoomed in part for a period of one hour or one hundred kilometres and one part for the whole day or 700 kilometres
- Zoom settings: auto zoom, manual and default settings

Meteo (weather maps)

- Display of generated weather maps:
 - Temperature
 - Humidity
 - Pressure
 - Wind speed
- Display of layers on the basic map:
 - Low clouds
 - Mid clouds
 - High clouds
 - Precipitation
 - Wind stream lines
- Automated detection of available weather models and available runtimes
- Display of satellite images: visible/infrared/radiation prediction
- Animated weather charts

SHOULD HAVE

- The application is usable in multiple formats

COULD HAVE

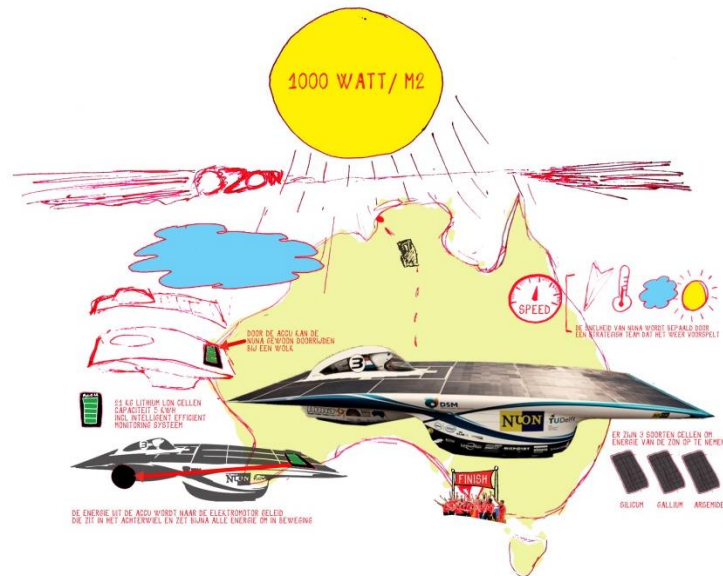
- Auto speed adaptations based on strategy file (on/off) in mission control panel
- Advised speed (km/h) in mission control panel
- MT-SAT prediction of radiation (W/m²) in meteo panel
- In case of an error message or warning, the application will notify the user and will create a shortcut to the status/temperature panel
- A scroll function for the map
- Satellite panel with
 - Diagrams of the measured solar radiation, wind speed, wind angle versus predicted values of models and runtimes
 - Diagrams of predicted values based on the most recent strategy
 - Real time verification of the weather models including statistics (mean deviation, standard deviation)

WOULD HAVE

- Instead of an adaptable config file for the properties of the solar car, a set-up file which automatically saves the car, weather and country preferences

BUILDING A GUI FOR NUNA'S MISSION CONTROL

RESEARCH PHASE REPORT



CLIENT

Rune van der Meijden

FORMER APPLICATION DESIGNER

Noud Brasjen

SOFTWARE DEVELOPMENT TEAM

Isha van Baar

Zakaria el Bakkali

Jurian van Dalfsen

Eveline van der Schrier

Lodewijck Vogelzang

TEAM COACH

Nicolas Dintzner

CONTACT

rune@nuonsolarteam.com

noud.nuna6@gmail.com

I.J.vanBaar@student.tudelft.nl

Z.elBakkali@student.tudelft.nl

J.vanDalfsen@student.tudelft.nl

E.D.vanderSchrier@student.tudelft.nl

L.G.J.Vogelzang@student.tudelft.nl

N.J.R.Dintzner@tudelft.nl

ABSTRACT

A large amount of work has been done in the research phase. Since the first main goal is to refactor an application, a code analysis and two class diagrams of the current application have been made. Then, after doing research on programming languages, GUI frameworks and real-time connection frameworks by comparing existing ones, the decisions fell on Java, JavaFX and Java.net.

Since the second main goal is to develop a new application, a structure of the new application has been made. This structure will be build according to a client-server model. Several models that each provide a different view on this new application have been made. These models will help the software team during the development phase but also provide a good overview for any future stakeholder that tries to update the system. The diagrams that have been made represent the high-level component and data logical flow diagram and the low-level server, client and GUI class diagrams. During the making of these models some important design choices have also been made, described in this report.

LIST OF CONTENTS

Abstract	2
Introduction	5
Current application	5
Reverse engineering	5
Problems at understanding current application	5
Research areas	6
Programming language	6
LabVIEW	7
Java	8
C#	8
Conclusion	9
GUI Frameworks	9
QT JamBi	9
Swing	9
JavaFX	9
Conclusion	9
Real-Time connection frameworks	10
Java.net	10
Javax.net	10
Conclusion	10
Modelling new application	10
High-Level design choices	10
Component diagram	10
Logical data flow diagram	11
Low-Level choices	12
Problems at designing new application	12
Conclusion	13
Appendices	14
Appendix I: Current application mission control code analysis	14
frmMain.frm	14
frmMainClient.frm	14
mdlArchive.bas	14
mdlData.mdl	14
mdlDataColor.bas	14
mdlDataMobiboxx.bas	14
mdlDataServer.bas	15
mdlGPS.bas	15
mdlMain.bas	15
mdlMainGraphics.bas	15
mdlNuna7.bas	15
mdlPanel.bas	16
mdlPanelBattery.bas	16

<u>mdlPanelMeteo.bas</u>	16
<u>mdlPanelMissionControl.bas</u>	16
<u>mdlPanelPowerFlow.bas</u>	17
<u>mdlPanelRoute.bas</u>	17
<u>mdlPanelSatellite.bas</u>	17
<u>mdlPanelStatus.bas</u>	17
<u>mdlPNG.bas</u>	17
<u>mdlPyranometer.bas</u>	17
<u>mdlThermometer.bas</u>	17
<u>mdlTopBar.bas</u>	17
<u>mdlWindmeter.bas</u>	17
<u>Appendix II: Current application strategy code analysis</u>	19
<u>frmMain.frm</u>	19
<u>mdlAdditionalGraphics.mdl</u>	19
<u>mdlAdditionalProcedures.mdl</u>	19
<u>mdlData.mdl</u>	19
<u>mdlDataServer.mdl</u>	19
<u>mdlDataStrategy.mdl</u>	19
<u>mdlMain.mdl</u>	19
<u>mdlMainGraphics.mdl</u>	19
<u>mdlPanelControl.mdl</u>	19
<u>mdlPanelStrategy.mdl</u>	20
<u>mdlSideBar.mdl</u>	20
<u>mdlStrategy.mdl</u>	20
<u>mdlStrategyOld.mdl</u>	20
<u>mdlTopBar.mdl</u>	20
<u>Appendix III: Current application - Class diagram - Mission control</u>	21
<u>Appendix IV: Current application - Class diagram - Strategy</u>	23
<u>Appendix V: New application – Component diagram</u>	25
<u>Appendix VI: New application – Logical data flow diagram</u>	26
<u>Appendix VII: New application - Class diagram - Server</u>	27
<u>Appendix VIII: New application - Class diagram - Client</u>	28
<u>Appendix VIII: New application - Class diagram - GUI</u>	29
<u>References</u>	31

INTRODUCTION

During three weeks, week 11, 12 and 13, the software development team has been performing research for the development phase. The research has been divided in two parts. The first part consists of researching what kind of application the software development team has to create. This part has been completed during week 11 and 12. During week 11 the team has completed the following tasks:

- Writing the Plan of Action document
- Creating a requirements list with user and system requirements
- Having meeting sessions with the client and others from the Nuna team
- Being able to run the current source code on our own computers
- Reverse engineering the architecture of the current application implementation

The second part of the research consists of researching how the software development team can create the new application as good as possible. The team researched if certain parts of the functionality of the application required additional frameworks and which frameworks are best suitable for these functionalities. The team also modelled the new application by creating models at different levels of the application. This part has been completed during week 12 and 13. During week 12 the team has completed the following tasks:

- Researching GUI frameworks in Java
- Researching real-time connection frameworks in Java
- Reverse engineering the architecture of the current application implementation

During week 13 the team has completed the following tasks:

- Modelling the new application

The results of the conducted research are documented in this report. In chapter 1 the findings about the old application are documented. Models and documentation of the old application have been written. In chapter 2 the results of the research into frameworks for the new application are documented. Chapter 3 consists of models for the new application.

CURRENT APPLICATION

The problems with the current application occur in the lack of extendibility and maintainability of the program. The current application is written in Visual Basic 6 which is a not properly supported programming language. Extending the current program by adding some functionality is a very difficult thing to do because it requires a good understanding of the current, badly designed system and also a good understanding of Visual Basic 6. Furthermore, there is no official documentation available and diagrams of the architecture have never been made. This is why the team needed to reverse engineer the current application.

REVERSE ENGINEERING

The current application consists of two main programs: *MissionControl* and *Strategy*. During the reverse engineering of the software development team, the team has tried to find the purpose of each Visual Basic 6 file (.mdl, .frm, .bas) and its contents, excluding the files and methods that clearly included mostly graphical data. The code analysis is given in in *Appendix I: Current application mission control code analysis* and *Appendix II: Current application strategy code analysis*.

After the code analysis, the team produced two automatically created class diagrams (see *Appendix III: Current application - Class diagram - Mission control* and *Appendix IV: Current application - Class diagram - Strategy*). The original idea was that these models could help the team with the structure of the new application. However, while these diagrams give a good overview in how each of the modules is composed and what operations can be used, they have not been used in the rest of the research phase. Yet they could still be of importance at the beginning of the development phase.

PROBLEMS AT UNDERSTANDING CURRENT APPLICATION

Some issues (marked in red in *Appendix I: Current application mission control code analysis* and *Appendix II: Current application strategy code analysis*) still remain after the code analysis as they involve extremely low level knowledge that could only be answered at the first meeting with the former application designer in the development phase.

Apart from other specific questions about the source code of the current application, the questions mainly concern the knowledge about which calculations are needed to obtain the correct values that need to be displayed in the new application. Thus after the analysis, it could be concluded that the following sensor data items still involve some unknown calculations:

- Battery state (Wh) plus CMS, BMS and model
- Battery power (W) plus CMS, BMS and model
- Power in (W) plus CMS, BMS and model
- Power out (W) plus CMS, BMS and model
- Actual radiation (W/m^2)
- Maximum and minimum radiation (W/m^2)
- Maximum and minimum cloud fraction (%)
- Deviation of cloud fraction (%)
- BSoC (V) plus BMS and model
- Maximum and minimum BSoC (V)
- Maximum and minimum battery temperature ($^{\circ}\text{C}$)
- Power per MPPT (W)
- Battery voltage (V)
- Maximum and minimum battery voltage (V)
- Current battery (A)
- Motor power (W)

RESEARCH AREAS

One of the important choices that the software development team had to make was deciding which programming language is going to be used. A consideration between different programming languages had to be made as choosing a language that is not suited for this project could mean that future problems will rise much easier.

Another important choice that the team had to make was which GUI framework should be used. The choice for the GUI framework is a very important one because the main goal is developing a GUI. Developing takes a lot of time and effort and the team wants to deliver a good product. Different frameworks have different possibilities and choosing the right one could be crucial for the product because the development phase is limited to a certain amount of time.

A real-time connection framework had to be chosen as well. Choosing a framework of this kind is an important matter because the data the application is using comes from a Wi-Fi connection. When this connection does not work (as it is supposed to) you could have a problem with the data not sent properly and therefore having no application input. This connection has to be secure and fast. Different frameworks with different possibilities exist in this field.

To conclude it is clear that in our application context, these three research areas are important:

- Programming language
- GUI Java Research
- Real-time Java Research

PROGRAMMING LANGUAGE

Looking at the knowledge gained from the current application and the *List of Requirements*, there are a few important criteria for the programming language of the new application:

- The programming language should support Windows 7 and 8. However, it would be very nice in regard to the application's robustness and extendibility if the language is easily adapted to other operating systems than Windows.
- The programming language should not require the user to install a lot of extra programs, packages, etcetera. This is especially true since we assume that the user does not necessarily have an understanding of how the application works.
- It should be easy for later software development teams to improve and/or extend the system. In order to suffice this condition, the language should be well documented. Moreover, it is preferable that the language is widely used, since this usually means that a lot of user tips, tutorials and examples are available on the Internet.
- It should be easy to implement a rich GUI
This means that the language should have access to at least one framework or library focussed on the creation of GUIs, preferably as many as possible. The created GUIs should support multiple panels and be easily extendable, customizable and testable. These and other requirements will be tested in the section *GUI Frameworks*.

- It is preferable that the majority of the software development team is already familiar with the language because of the limited amount of time.

Three languages that might fit these requirements well will now be discussed. The languages are chosen based on an initial pre-research on the Internet, own experiences and a company's presentation which was recommended by one of the members of the Nuna Solar Team. Also, languages that are meant for Internet usage like HTML, Javascript, etcetera were disqualified as they could only be executed in browsers.

LABVIEW

LabVIEW is a graphical programming environment. It is designed by the National Instruments company. One of the advantages to programming with LabVIEW is that its source code is very similar to circuit diagrams, making it a very easy language for electrical engineers to pick up. That is why LabVIEW has become popular in the data acquisition and automation fields, where there are plenty of electrical engineers.

Several team members joined a presentation from National Instruments. This presentation was specifically organised for all the so-called Delft Dream Teams, teams which realize extremely advanced machines like the Nuna Solar Car. It became clear that National Instruments offers their expensive products for free to students of Delft University of Technology.

After asking some more questions, the team found out several factors that could make LabVIEW an ideal language for our application. Firstly the graphical programming of the LabVIEW environment where GUIs are simply assembled from a palette of available front panel controls, should create an easy and quick way to implement a solid GUI. Also, LabVIEW supports real-time connections as TCP/IP, which could be important in regard to the real-time connection framework of our application (as seen in the *Real-Time connection frameworks* section).

A third advantage is LabVIEW's compatibility with hardware. National Instrument's main business remains data acquisition hardware and therefore they will ensure that LabVIEW will always instantly be able to communicate with their hardware. As the Nuna team might change their data acquisition hardware and Mobiboxx in the future, it could be very well possible that National Instruments will be chosen. Other Solar Car Teams already use National Instrument's software.

All of these reasons appear to form a good reason to choose LabVIEW as the new application's language. Unfortunately though, there are a lot of problems. Although the language supports Windows 7 and 8 and does not require the user to install various extra programs, it does require future software development teams to have access to LabVIEW. This is not just a language, it is a programming environment and the LabVIEW language could thus only be accessed in the corresponding IDE. Thus future access to LabVIEW is not completely assured which hugely decreases the extendibility of applications in this language.

Moreover, although LabVIEW seems to have plenty of tutorials and documentation, the opinions of professional software developers who do not use the program solely for electrical engineering purposes reveal a lot of critical problems. For instance, LabVIEW does not seem to have the functionality of writing descriptive comments, naming variables. But most of all, its graphical programming interface seems to quickly turn into a web of wires. This makes programs messy, hard-to-follow and impossible to debug. The following LabVIEW instance is described as easy on Wikipedia.

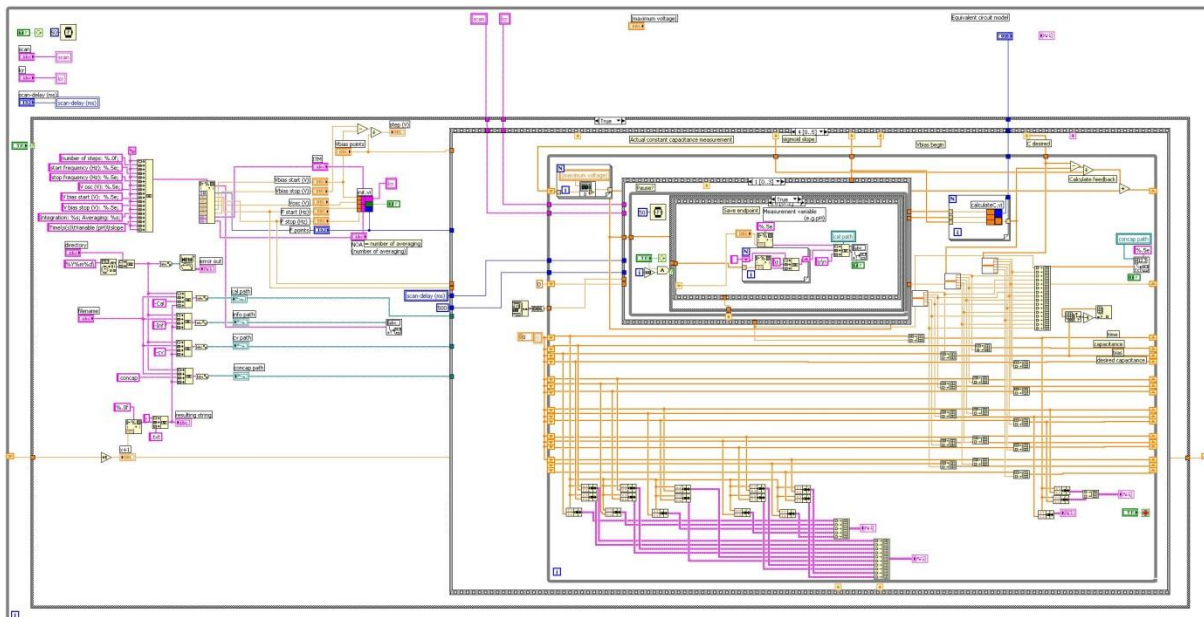


Figure 1: Example of an 'easy' LabVIEW instance [2]

JAVA

Java is a widely used object-oriented programming language developed by Sun Microsystems. It is based on C++ but has a more extensive class library. Java is a very powerful language. Its main advantage is that it is written to be platform-independent. This means that Java could not only run on required platforms as Windows, but also on Mac OS. Also, the language is not completely dependent upon one IDE such as the LabVIEW or Visual Studio IDE, but can be executed in multiple IDEs. Together with its massive support and documentation, applications written in this language are the most 'open', robust and extendable of all three. The main requirement of our new application of easily improvable and extendable system could thus very likely be sufficed with Java.

For the users it is also easy to run a Java program. If not already present on their computer, they only need to install the Java 7.

As Java has access to a lot of internal and external GUI libraries, it will be very likely that there is at least one way to implement a rich GUI. Even graphical oriented implementing a GUI structure are available. A possible disadvantage lies in the fact that these libraries might not all be that clear, documented, and adapted to Java. Thus the very big advantage of Java, its platform independency, might also result in adequate but not great GUIs.

Lastly all five members of our software development team are very experienced in Java so the coding could be started right from the beginning of the development phase. This saves a lot of time and unnecessary helpless efforts.

C

C# is in many ways a similar to Java. This also follows from the fact that the language is designed by Microsoft as an answer to the popular Java language. Although C# has a smaller library, it is also object-oriented and in syntax and semantics similar to Java. Also, just like Java, C# has automatic memory management, which frees software developers from manually allocating and freeing memory occupied by objects by the presence of a garbage collector.

However, this also forms one of the main drawbacks as C# files could only be executed in Microsoft's Visual Studio and on Microsoft platforms. Thus it is slightly more 'closed' and exclusive, giving a bigger chance of becoming Microsoft's next Visual Basic 6 and therefore becoming less convenient for future software development teams.

On the other hand, many professional software developers think that C# is just as robust and effective as Java. Also, in many occasions, C# could replace the client or server with Java as its counterpart.

Overall, it could be concluded that C# has both advantages and disadvantages and its use seems to be more based on personal preferences. However, as one of the software team members has had bad C# experiences in a previous project and all members have more experience with Java, this might be just the inducement that Java is preferred over C#.

CONCLUSION

The conclusion the software development team has drawn is to choose for Java. Java suffices all of the proposed requirements in the most proper way. Although it does not differ that much from C#, personal preferences and experience form just the difference to decide to go for Java.

GUI FRAMEWORKS

The application will be used during races with the Nuna Solar Car. All information about the solar car and the sensors is displayed in the application so that the best strategy can be created during the race. Therefore it is important that all data is shown in a clear way so that during the race the strategists have a good overview of the data they want.

The software development team has researched multiple framework for a GUI in Java. The framework has to be able to create a GUI that is specifically good for this application, it needs to be able to offer all functionality in the GUI that is required for the application. This functionality includes support for:

- Multiple panels
- Easily extendable
- Customizable
- Testable
- Fullscreen support
- Scalable GUI support

The software development team researched the following frameworks: Qt Jambi, Swing and JavaFX.

QT JAMBI

Qt Jambi is a framework created for Java using the native *Qt* library for C++. The framework offers the required functionality for the GUI, however the framework is not very intuitive and creating a GUI with some basic elements was hard to do during the tests of this framework. For instance creating a window with two panels, one on the left half of the window, one on the right half of the window required a lot of *QWidget* and *layout* objects. The code required to implement this simple GUI was too extensive. If this framework is used for the application, the software development team will need a lot of time to create the required GUI for the application. Since the code for the GUI will be very extensive, it will be hard for future software development teams to extend the GUI. Therefore the *Qt Jambi* framework can be used as the GUI framework for the application, but has some severe disadvantages.

SWING

Swing is the Java build-in framework for creating GUIs. It is therefore used a lot, what is good because solutions to common problems are easy to find on the Internet. Another advantage is that the documentation for SWING is complete and in a simple format. However, many other GUI frameworks for Java are invented because of one simple reason: SWING is difficult to use. It is difficult to create a fluid layout and to customise standard GUI components. A newer and widely used GUI framework for Java is JavaFX.

JAVAFX

JavaFX is made by Java/Oracle to simplify making a GUI. Because Oracle, the owner of Java, made it, JavaFX is also well documented just like SWING. In contrast to SWING, this framework is not built-in for Java. Therefore it is slightly more difficult to implement this in a Java project. Some installations are required and imports of packages in the code are needed. This is a disadvantage to both the current development team and later development teams. But when this frameworks is set up, it makes it easier to create a GUI. Especially in combination with a 'scene builder', a *What-You-See-Is-What-You-Get* application to create GUIs without codes. It is easy to create a basic layout in this application and with addition of CSS it is also very easy to customise the components like buttons and texts.

CONCLUSION

The frameworks are compared and JavaFX is found to be the best option for this project. The most important part was the ability to customise the GUI easily. It is clear that the new application should look good, like the current application. Because JavaFX supports the use of CSS is JavaFX the best to work

with. The greatest disadvantage is that it might be difficult to improve/extend the application for later development teams. Guidelines are going to be created to minimise this problem.

REAL-TIME CONNECTION FRAMEWORKS

The application gets all data of the Nuna Solar Car from a Wi-Fi connection with the Mobiboxx and the sensors. It is crucial for the application that all data from the Nuna Solar Car is displayed as soon as possible. Therefore it is important that the application can send and receive data over the connection fast and without problems. Since only one application (the server) can be connected to the Mobiboxx at the same time, the information needs to be distributed to all available running applications (the clients). The real-time connection framework needs to be able to execute two tasks:

- Read data from the connection with the Mobiboxx
- Send the data to other available applications

The software development team has researched available frameworks in Java for this task. The researched frameworks are Java.net and Javax.net.

JAVA.NET

Java.net is the standard framework for networking in Java. The package includes the *Socket* class which can start a connection to the Mobiboxx by specifying the IP-address and port of the Mobiboxx. The *Socket* class offers the functionality to create an *InputStream* object and an *OutputStream* object. With these objects the application can read from the connection and write on the connection. This class offers the functionality needed for the connection to the Mobiboxx.

The package also includes the *SocketServer* class. With this object the application can function as a server. All other applications can open a socket to this server. The server application can easily distribute all data to other available applications.

Java.net offers the functionality to execute the two required tasks. Therefore the Java.net framework is a good solution as the real-time connection framework.

JAVAX.NET

The framework Javax.net is an extension of the Java.net framework. The Javax.net framework also offers the functionality of the *Socket* and the *ServerSocket* class. Additionally the Javax.net also offers the functionality to create secure sockets with the *SSLSocket* class and the *SSLServerSocket* class. Therefore the Javax.net is also a good solution as a real-time connection framework.

CONCLUSION

Javax.net is the best option for the real-time connection. This package offers the functionality required for the connections and additionally offers secure connections. Therefore the software development team has chosen to use the Javax.net framework for the real-time connections.

MODELLING NEW APPLICATION

HIGH-LEVEL DESIGN CHOICES

To create the models the software development team has made difficult decisions about the architecture of the application. In this paragraph will be explained which decisions the team has made for the design of the application.

COMPONENT DIAGRAM

The most important design choice made by the team is deciding the components the application consists of and what their responsibilities are. The team has chosen for the following components and their responsibility in the Server application:

- Controller: Directing the application
- SensorConnection: Handling the connection with the sensors
- ClientConnection: Handling the connection with the clients
- DataManagement: Data conversion and saving data in a repository

The team has chosen for the following components and their responsibility in the Client application:

- Controller: Directing the application
- GUI: Visual representation of the data
- ClientConnection: Handling the connection with the server
- DataManagement: Data conversion and saving data in a repository

The applications and their components are visualised in the component diagram, *Appendix V: New application – Component diagram*.

LOGICAL DATA FLOW DIAGRAM

The component diagram is a high level visualisation of the architecture. However, to create a class model, a more detailed model is necessary with specific tasks for each component. To decide the specific responsibilities of each component, the team first made decisions about certain parts of the architecture. The following paragraphs will be about the important choices the team had to make to create a consistent data flow.

One of the basic choices the team had to make was deciding whether the calculations with the data from the Mobiboxx are done on the server or the client. The main advantage of the client calculating the necessary data is that clients are not dependent on the server data. For instance, in case of a server crash the average speed over the last fifteen minutes cannot be calculated for the next fifteen minutes. The main advantage of the server calculating all the data is the fact that all calculations only have to be executed once. Also, since the value for the average speed over the last fifteen minutes will be calculated in the server, the client will immediately have this value without having to collect data for fifteen minutes to obtain this value. The team has chosen for executing calculations on the server and logging all the data the server calculated. Therefore the calculations only have to be executed once and when the server has to be restarted, the server can use the log data for values like the average speed over the last fifteen minutes.

After deciding that the calculations are executed by the server, the team had to make sure the server application would be able to safely send the data to the client. Therefore every byte array received by the client is checked for completeness by checking the header of the data stream. If a header is missing or the byte array does not have the expected length, the client will not process this byte array. By not processing the incomplete byte arrays, all processed data should be complete. This design choice could be improved by not completely deleting incomplete byte arrays, but rather separately process all parts of the array, where each part is the data from one sensor, or by assigning the value -9999 to the missing values. The team will not initially implement these improvements, but rather focus on the main functionality of the application.

After the invalid data is deleted, the team had to decide how the application will transform the byte array from the Mobiboxx to the data objects of DataManagement. One possibility is to read the data from the byte array and store the values in temporary data objects. Then the data in the temporary data object is used for the calculations and saved in the data objects of DataManagement. The advantage of this possibility is that the process is organised and easy readable in the code. The second possibility is to read the data from the byte array and immediately use it for the calculations and store it in DataManagement. The main advantage is that this solution requires less storage space than the first solution since it does not need temporary data object to save data. The disadvantage is that the code might be unclear. The team has decided to use the second solution and will pay extra attention to the readability of the code for this process to keep the code clear.

One of the requirements of the application (see *Plan of Action* document) is that all data should be logged. Since logging data multiple times is not desirable, the team decided to only log the data stored in the data objects after it is processed. This contains all the data available from external sources and the data calculated from this data. Therefore logging at this point in the application will log all data at the server side. Since the client only displays data it receives from the server, additional logs in the client are not necessary.

The team also had to decide when the server would send data to the client. This could be done using a *push model* where the server sends the data to the client whenever new data is available. The advantage of this model is that every time new data is available, it is immediately sent to the clients. The other option is to send data according to the *pull model* where the client requests the data it needs in the GUI every unit of time. The advantage of this model is that the server only needs to send the data that is needed for the panels visible in the GUI. The team has chosen for the *push model* to reduce the time needed to display data in the GUI and therefore enables the user to get more accurate data.

One of the concerns of saving all the data in data objects at both the client and the server side was the amount of objects that will be created. Since at one point in time there are thirteen objects containing the current data and since thirteen new objects are created each time new data is

available, the costs for these objects may increase very fast. However, old data objects will not be stored in the application. Since Java has a proper garbage collector, these objects will be destroyed. The team therefore decided to not save any of the old data in data objects and therefore not create a huge amount of objects.

The application has to be easy extendable (see *Plan of Action* document) and this has to be acquired by modelling the application the right way. The different components have been chosen to ensure this. By dividing the application into different components that each are responsible for certain tasks, future developers can easily change the way one of the components works. As long as the functions provided by a component still provide the same functionality, the actual implementation of the function does not matter. Also, when the application has to be extended (i.e. adding a sensor to the sensors of the chase vehicle), this structure will make it clear where code has to be added for the program to work with this new sensor.

The team was however concerned with the amount of data that multiplied within the application. For instance, when the user sets the cruise control speed in the application, the GUI component will create an object for this action. This object is sent to the controller, which sends it to the DataManagement component, which in its turn sends it to the Connection component. Therefore this object is available in four different components. To avoid the creation of four different objects for one action, the architecture has been designed to send the objects to other components by calling functions in these components. By using this method, only pointers to the actual object are sent between the different components and no extra objects are created.

After the team made decisions about the detailed responsibilities of each component, the logical data flow architecture was created, containing the possible data flows in the application and therefore the specific responsibilities of each component. The logical data flow diagram can be found in *Appendix VI: New application – Logical data flow diagram*.

LOW-LEVEL CHOICES

After the design of these high-level diagrams, an initial idea of the low-level architecture could be made. Thus it was decided to construct a rough overview of the components, classes, attributes and methods of the new application. This is visualized in three class diagrams, available in *Appendix VII: New application - Class diagram - Server*, *Appendix VIII: New application - Class diagram - Client* and *Appendix VIII: New application - Class diagram - GUI*.

From these diagrams, it becomes clear that the components from the component diagram (*Appendix V: New application – Component diagram*) now form separate packages. Also, from the methods it is clear that the logical data flow as (*Appendix VI: New application – Logical data flow diagram*) will be followed. The third class diagram of the GUI component (*Appendix VIII: New application - Class diagram - GUI*) still consists of major gaps and issues. However, this diagram will only be definitive after the first few sprints, when the team has found its way in the GUI library of JavaFX.

PROBLEMS AT DESIGNING NEW APPLICATION

There were several technical problems the development team ran into. Most of the problems were created because of a lack of specific knowledge. Luckily, a solution was found for almost all problems or questions.

Problem/question	Solution
Strategy panel is twice as big as other panels (like meteo panel or overview panel)	If strategy panel is selected, it takes space for 2 panels or a full screen strategy panel pop up is created
How can the COM-port id of the sensors dynamically be detected?	LJM package of LabJack can help with that.
How can a socket setup a connection to a server socket without knowing its IP address?	Use local host.
We want a global accessible variable for the setup file location. What is the best way to implement this?	Create a class with a static variable.
Which problems/errors can occur while sending a byte array over Wi-Fi? (And in cabled network?)	-
What is the easiest/best way to create an executable file from Java code?	Use eclipse to create an executable JAR file.
Current sensors of chase car are connected via a	-

serial port. Is it possible to change it to USB? Does it affect the code?
--

Table 1: Technical problems

CONCLUSION

The problems in the current application of the Nuna chase car lie in the lack of extendibility and maintainability of the program. Therefore the team has the task to create and develop a new application, and this time one that is easily extendable and maintainable with good documentation and diagrams of the architecture (see *Plan of Action*).

Firstly the current application has been reverse engineered by analysing its code and constructing two class diagrams. Several problems involving calculations on values have been noted.

With the previously proposed requirements (*List of Requirements*) in mind, some choices regarding the programming language and its frameworks have been made. Firstly, the software development team has chosen to use the Java programming language. Java suffices all of the proposed requirements in the most proper way. Although it does not differ that much from C#, personal preferences and experience form just the difference to decide to go for Java.

Secondly some frameworks to construct Graphical User Interfaces (GUI's) have been compared, knowing that the application will be implemented in Java, and JavaFX has found to be the best option for this project because it meets all the proposed requirements.

Thirdly the team has chosen to implement the application in a client-server model. The server will be connected to the Mobiboxx and the chase car sensors. It will calculate the data needed for the clients. After calculating the necessary data, these will be logged and are available to the clients. By choosing for this architecture the client will have very little responsibilities. The communication between the server and the client will be using the push model. When the server has new data available, this data will immediately be send to all clients. Incomplete byte arrays will be deleted until the team has time to find a better solution. This way the risk of invalid data will be reduced.

After the performed research, the new application has been modelled in terms of diagrams. A component diagram has been made to get a high-level vision on the new application and a logical data flow diagram has been made to determine the components' responsibilities. Lastly low-level class-diagrams have been made to visualise the dependencies and the low-level structure.

APPENDICES

APPENDIX I: CURRENT APPLICATION MISSION CONTROL CODE ANALYSIS

FRMMAIN.FRM

GUI bestand.

FRMMAINCLIENT.FRM

GUI bestand. Precies hetzelfde als hierboven, behalve resolutie en gecommente stukken.

MDLARCHIVE.BAS

GUI bestand van Nuna6. Wat er precies wordt gedaan met de oude gegevens zoals het oude BMSPanel is niet duidelijk.

MDLDATA.MDL

Dit bestand bevat een aantal algemene functies.

Conversion function: AscToDate, DateToAsc, AscToFloat, FloatToAsc, AscToBinary, ByteToBinary, ByteArrayToString.

BMSModuleIndex: retourneert index van BMS module van input battery cell index.

BMSCellTemperature: leest temperatuur van input cell index uit van dataBMS.Temperature.

MotorLimitFlag, MotorErrorFlag, BMSTaskFlag, BMSErrorFlag, MPPTErrorFlag en ConnectionState: retourneert string van bijbehorende index.

MDLDATACOLOR.BAS

GUI bestand, bepaalt de kleuren van cijfers.

MDLDATAMOBIBOXX.BAS

Het doel van dit bestand is het onderhouden van de connectie tussen de server en de Mobiboxx en het verwerken van de data.

Attributenlijst: lijst met alle belangrijke attributen die kunnen worden ontvangen, bijna allemaal Strings.

LoadMobiBoxx: voorbereiding op het aanmelden server bij sensoren via het standaard bericht + specifieke code voor de sensor. Daarbij worden standaardwaardes geset met characters en een txt-bestandje weggeschreven.

ConnectMobiBoxx: aanmelden van de server bij sensoren.

DisconnectMobiBoxx: afmelden van de server bij sensoren.

GetMobiBoxxData: data opvragen uit de mobiboxx. Belangrijke connection gegevens.

GetMissionControlData: data opvragen uit de volgauto (GPS, wind, ...)

ProcessMobiBoxxData: uitlezen van de verkregen data uit de mobiboxx en dat op een goede manier opslaan.

CheckLogFilesExistance: kijk of de logbestandjes voor elke sensor bestaan. Wij moeten die logbestandjes nog even krijgen.

UpdataLastMinuteData: update data met behulp van gegeven tijdsintervallen. Erg belangrijk. Gedaan voor Speed, BSoC, PowerIn, PowerOut, EffectiveWind, CloudFraction, Meteo.

LoadStrategy: de strategiefles van Matlab worden ingeladen om de strategiedata te vormen.

LoadStrategyRun: hetzelfde als hierboven. Geen idee wat het verschil is, vragen.

MDLDATASERVER.BAS

Het doel van dit bestand is het onderhouden van de connectie tussen de server en de client en het loggen van alle verkregen data in logfiles.

LoadMissionControlServer: sluit de oude verbinding en luistert en opent een nieuwe verbinding (de client doet dit).

DisconnectMissionControlServer: de client disconnect van server.

SendMissionControlServerData: de server verstuurt alle data.

LoadMissionControlClient: de server start verbindingen op port 6000 tot 6009. Niet duidelijk waarom die poorten.

DisconnectMissionControlClient: de server sluit alle verbindingen.

GetMissionControlServerData: haal data op van de server.

ProcessMissionControlServerRawData: zet de data van de server (die de server van de Mobiboxx gekregen heeft) in objecten. Schrijf alle data naar de logfiles.

ProcessMissionControlServerProcessedData: zet de data van de server (die de server verwerkt heeft vanuit de Mobiboxx data) in objecten. Schrijf alle data naar de logfiles.

MDLGPS.BAS

Het doel van dit bestand is het ontvangen van de GPS data.

LoadGPS: laadt de GPS door de poort goed te zetten.

DisconnectGPS: sluit de poort.

GetGPS: verwerk de data van de GPS meteo sensor met bepaalde berekeningen voor longitude en latitude en zet de data in het bijbehorende object.

MDLMAIN.BAS

GUI bestand.

SetSpeed: GUI methode (dblSpeed attribuut wordt geset, maar is niet de snelheid van de auto). Wat is dblSpeed?

MDLMAINGRAPHICS.BAS

GUI bestand.

MDLNUNA7.BAS

Het doel van dit bestand is omzettingen te doen die nodig zijn voor de gegevens in het Meteopanel + delen van het MissionControlpanel (PowerIn, PowerOut, BSoC).

DrawNuna7: GUI function. Retourneert een gedraaid plaatje van de Nuna voor Meteopanel.

ShortwaveRadiationToPowerIn: ShortwaveRadiation wordt omgezet naar PowerIn met een bepaalde natuurkundige formule ($x \times 6 \times 0.17$).

PowerInToShortwaveRadiation: PowerIn wordt omgezet naar ShortwaveRadiation met een bepaalde natuurkundige formule, het omgekeerde van hierboven.

WindToPowerOut: met de Speed, EffectiveWindspeed, EffectiveWindDirection en 6 integers a t/m f wordt dePowerOut berekend in een lastige natuurkundige formule.

PowerOutToWind: met de Speed en PowerOut wordt de Wind berekend in een lastige formule, anders dan hierboven.

WindmeterToEffectiveWind: met de WindSpeed, WindDirection en Speed wordt de EffectiveWind berekend in een formule.

WindmeterToWindSpeed: met de WindSpeed, WindDirection, Speed en Heading wordt de WindSpeed berekend in een formule.

WindmeterToWindDirection: met de WindSpeed, (oude? tijdelijke raw data?) WindDirection, Speed en Heading wordt de WindDirection berekend in een formule.

WindmeterToWindDirectionUncertainty: met de WindSpeed, WindDirection, Speed en Heading wordt de WindDirectionUncertainty berekend na het gemiddelde van drie berekeningen..

WindmeterToWindComponents: met de WindSpeed, WindDirection, Speed en Heading worden de WindComponents berekend, namelijk vector WindU en WindV. Dit gebeurt in een formule.

WindComponentsToWindDirection: met de WindU, WindV, Speed, en Heading wordt de WindDirection bepaald in een formule, het omgekeerde van hierboven.

ShortwaveRadiationToCloudFraction: met de ShortwaveRadiation, Latitude, Longitude, en Time wordt de CloudFraction berekend. De CloudFraction wordt gezet op 0 of 1, afhankelijk van of de waarde negatief of positief is. Voor de rest: onbekende formule.

CloudFractionToShortwaveRadiation: met de CloudFraction, Latitude, Longitude en Time wordt de ShortwaveRadiation berekend met een omgekeerde formule t.o.v. hierboven,

SolarAltitude: met de Latitude, Longitude en Time worden met heel veel omzettingen (Day, JulDay, Anomaly, Center, EclLong, Ascension, Declination, SidTime, HourAngle) de zonnehoogte (SolarAltitude) berekend.

RadiationStatistics: met RadiationArray (array met allemaal waardes van de straling), RadiationMax, RadiationMin, en CloudFraction worden de Shortwave Radiation Maximum en Minimum en (Cloud) Fraction waardes berekend. Erg lange Sub met veel Statistics die berekend worden.

AltitudeToEnergyOut: met de AltitudeDifference wordt de EnergyOut berekend met een formule.

VoltageToBSoC: met Voltageln, Currentln, (oude? tijdelijke?) BSoCMean en BSoCStd wordt de BSoCmean berekend met een formule.

MDLPANEL.BAS
GUI bestand.

MDLPANELBATTERY.BAS
GUI bestand voor alle gegevens in het BatteryPanel.

MDLPANELMETEO.BAS
GUI bestand voor alle gegevens in het MeteoPanel.

Wiskundige functie Average.

DrawShortwaveRadiationStatistics: het balkje op basis van radiationstatistieken wordt getekend.

DrawCompass: compass wordt getekend.

MDLPANELMISSIONCONTROL.BAS
GUI bestand voor alle gegevens in het MissionControlPanel.

TargetSpeedSlider: GUI gedeelte van de cruise control slider. Zou handig kunnen zijn.

Wiskundige functies: *Average* en *Gradient*.

MDLPANELPOWERFLOW.BAS
GUI bestand.

MDLPANELROUTE.BAS
GUI bestand.

MDLPANELSATELLITE.BAS
GUI bestand.

LonToX: een mapping van gps coördinaten?

LatToY: een mapping van gps coördinaten?

LoadMTSAT: het laden van verschillende versies van de satellietbeelden (infrarood, real-time).

MDLPANELSTATUS.BAS
GUI bestand.

MDLPNG.BAS
Dit bestandje is bedoeld voor het manipuleren van PNG plaatjes, maar erg veel functies bevat die alleen bedoeld zijn voor VB6, waarschijnlijk niet door Noud geschreven.

MDLPYRANOMETER.BAS
Bestandje bedoeld om de pyranometer(radiatie) uit te lezen.

LoadPyranometer: ?

FunctionGetPyranometer: ?

MDLTHERMOMETER.BAS
Bestandje bedoeld om de thermometer(temperatuur) uit te lezen.

LoadThermometer: start thermometer en begin te uit te lezen.

DisconnectThermometer: stop met uitlezen van de temperatuur.

MDLTOPBAR.BAS
GUI bestand.

LatLonToDistance: berekent afgelegde/af te leggen afstand met behulp van GPS-coördinaten.

LatLonToLocation: berekent locatie met behulp van GPS-coördinaten.

ControlStopDistance: berekent afstand naar de eerstvolgende control-stop.

ControlStopDistanceString: retourneert een afstand naar de eerstvolgende control-stop in stringvorm.

ControlStopLocation: berekent locatie van de eerstvolgende control-stop.

MDLWINDMETER.BAS
Bestandje bedoeld om de windmeter uit te lezen.

LoadWindmeter: bepaalt de poort voor de connectie met de windmeter.

DisconnectWindmeter: sluit de poort van de connectie.

GetWindMeter: leest de windsnelheid/windrichting van dit moment uit.

APPENDIX II: CURRENT APPLICATION STRATEGY CODE ANALYSIS

FRMMAIN.FRM
GUI bestand.

sckMissionControl_DataArrival: roept aan *ProcessMissionControlServerProcessData*, methode om een inkomende string in objecten om te zetten.

sckMissionControl_Error: print error.

tmrTimer_Timer: haal de data van de *sckMissionControl* en refresh deze data in de GUI (elke 5 ticks).

MDLADDITIONALGRAPHICS.MDL
GUI-bestand.

MDLADDITIONALPROCEDURES.MDL
Tekent het Strategy-scherm en updatet de informatie.

RegGetValue: leest een waarde uit de registry en retourneert deze waarde.

UTCNow: retourneert huidige datum en tijd.

CopyURLToString: haalt de pagina van een url op en slaat deze op in een string of bestand.

Wiskundige functies: min, max, ceiling, frac, round, roundstr, Modl, Asin, Acos, Atn2, Sinh, Cosh, Tanh, Log10.

EarthDistance: berekenen van de afstand tussen twee coördinaten.

DateString: retourneert een datum in een bepaalde format.

MonthName, *WeekDayName*: retourneert string van bijbehorende index (1 is zondag etc.).

Order: sorteeralgoritme.

Group: elementen van input array worden afgerond of veranderd in een breuk, optioneel gesorteerd.

Array functies: *MeanArray*, *MaxArray*, *MinArray*, *ReverseArray*.

MDLDATA.MDL
Zie Mobiboxx/mdlData.mdl.

MDLDATASERVER.MDL
Zie Mobiboxx/mdlDataServer.mdl.

MDLDATASTRATEGY.MDL
Het doel van dit bestand is het retourneren van waardes van een bijbehorende index.

ColorIndexToColor, *AverageToInteger*, *AverageToString*: retourneert waarde van bijbehorende index.

MDLMAIN.MDL
Zie Mobiboxx/mdlMain.mdl.

MDLMAINGRAPHICS.MDL
Zie Mobiboxx/mdlMainGraphics.mdl.

MDLPANELCONTROL.MDL

GUI bestand.

MDLPANELSTRATEGY.MDL

GUI bestand.

LoadDiagramOptions: laadt de diagramopties.

UpdateDiagramOptions: updatet de diagramopties.

UpdateDiagram, *UpdateSingleDiagram*: updatet diagrammen.

DiagramStepNumber: berekent aantal stappen voor de x-as.

DiagramStepTime, *DiagramStepDistance*: berekent de stapgrootte voor de x-as.

MDLSIDEBAR.MDL

GUI bestand.

MDLSTRATEGY.MDL

Strategiebestand.

LoadStrategy: laadt het tekstbestand met de strategie in.

CheckStrategy: controleert of er een geldig strategiebestand is.

LoadStrategyRun: het lezen van het strategiebestand (data in object).

LoadObservations: ?

CheckLogFilesExistence: controleer of geldige logbestanden bestaan.

LoadConfigurationFile: laadt een configuratiefile in.

SaveConfigurationFile: slaat een configuratiefile op.

MDLSTRATEGYOLD.MDL

Oud strategiebestand.

MDLTOPBAR.MDL

Zie Mobiboxx/mdlTopBar.mdl.

APPENDIX III: CURRENT APPLICATION - CLASS DIAGRAM - MISSION CONTROL



```

<<Form>>
frmMain
+boolMouseDown: Boolean
+intXMouseDown: Double
+intYMouseDown: Double
+arrXMouseDown: Double
+arrYMouseDown: Double
+intXMouseDown: Double
+intYMouseDown: Double
+intXMouseOver: Double
+intYMouseOver: Double
+dataMeteorReferenceTemp: MeteorReferenceD...
+Form_KeyDown(KeyCode: Integer, Shift: Inte...
+Form_Load()
+Form_Resize()
+Form_Unload(Cancel: Integer)
+picMain_MouseDown(Button: Integer, Shift: In...
+picMain_MouseMove(Button: Integer, Shift: In...
+picMain_MouseUp(Button: Integer, Shift: Inte...
+picDialog_MouseDown(Button: Integer, Shift: ...
+picDialog_MouseMove(Button: Integer, Shift: I...
+picDialog_MouseUp(Button: Integer, Shift: Int...
+picTopBar_MouseMove(Button: Integer, Shift: ...
+sockMissionControl_ConnectionRequest(Index: ...
+sockMissionControl_DataArrival(Index: Integer, ...
+sockMissionControl_Error(Index: Integer, Numb...
+sockMissionControl_SendComplete(Index: Inte...
+sockMobiBoxx_Connect()
+sockMobiBoxx_DataArrival(bytesTotal: Long)
+sockMobiBoxx_Error(Number: Integer, Descripti...
+sockMobiBoxx_SendComplete()
+mrTimer_Timer()
+txtDisplayMessage_KeyDown(KeyCode: Integer, ...
+txtDisplayMessage_LostFocus()
+txtMeteorEdit_KeyDown(KeyCode: Integer, Shi...
+txtMeteorEdit_LostFocus()

```

```

<<Module>>
mdlPNG
+Function
+Token: Long
+gdipInit: GdiplusStartupInput
+Sub
+hDC: Long
+hBitmap: Long
+img: Long
+hBrush: Long
+Graphics: Long
+OrWidth: Long
+OrHeight: Long
+OrRatio: Double
+DesRatio: Double
+DestX: Long
+DestY: Long
+DestWidth: Long
+DestHeight: Long
+ID_IDispatch: GUID
+Pic: PICTURE
+IPic: IPicture
+WMFHeader: wmfPlaceableFileHeader

```

```

<<Module>>
mdlGPS
+LoadGPS(com...
+DisconnectGPS...
+GetGPS(comG...

```

```

<<Module>>
mdlThermometer
+LoadThermometer(comT...
+DisconnectThermometer...
+GetThermometer(comTh...

```

```

<<Module>>
mdlWindmeter
+LoadWindmeter(comW...
+DisconnectWindmeter(...
+GetWindmeter(comWi...

```

```

<<Module>>
mdlData
+dblMSec: Double
+strHex: String
+intSign: Integer
+intExponent: Integer
+dblFraction: Double

```

```

<<Module>>
mdlMainGraphics
+strDialog: String
+clr: Integer
+clrG: Integer
+clrB: Integer
+clrFR: Integer
+clrFG: Integer
+clRFB: Integer
+clRB: Integer
+clRBG: Integer
+clRGB: Integer
+intFactor: Double
+dblLeft: Double
+dblTop: Double
+dblWidth: Double
+dblHeight: Double
+foreColor: Long
+textColor: Long
+intDimH: Integer

```

```

<<Module>>
mdlPanelStatus
+intDimW: Integer
+intDimH: Integer
+LoadStatusPanel(Pan...
+UpdateStatusPanel(Pan...
+UpdateBatteryStatusPanel(...
+UpdateMotorStatusPanel(o...
+UpdateMPPTStatusPanel(o...
+DrawTopShell(Left: Double...
+DrawUnderShell(Left: Double...

```

```

<<Module>>
mdlPanelMissionControl
+optRemoteControl: SwitchOptions
+optAutoSpeed: SwitchOptions
+optSpeed: FieldOptions
+optTargetSpeed: FieldOptions
+optSpeedLastMinute: FieldOptions
+optSpeedDeviation: FieldOptions
+boolSpeedAll: Boolean
+boolSpeedLastMinuteAll: Boolean
+boolSpeedDeviationAll: Boolean
+optBSOC: FieldOptions
+optBSOCLastMinute: FieldOptions
+boolBSOCAll: Boolean
+boolBSOCLastMinuteAll: Boolean
+optPowerIn: FieldOptions
+optPowerInLastMinute: FieldOptions
+boolPowerInAll: Boolean
+boolPowerInLastMinuteAll: Boolean
+optPowerOut: FieldOptions
+optPowerOutLastMinute: FieldOptions
+boolPowerOutAll: Boolean
+boolPowerOutLastMinuteAll: Boolean
+optDisplayMessage: FieldOptions
+numCountConnection: Byte
+dblTargetSpeedDialog: Double
+boolDisplayMessage: Boolean
+boolBMS: Boolean
+intDimW: Integer
+intDimH: Integer
+LoadMissionControlPanel(Pan...
+UpdateMissionControlPanel(Pan...
+TargetSpeedDialog(picOutput: Pict...
+TargetSpeedSlider(numDigit: Integer...
+DisplayMessageShow(txtDisplayMe...
+AveragedblInput: Double: Double
+Gradient(dblInput: Double): Double

```

```

<<Module>>
mdlPanelBattery
+optBatteryBSOC: FieldOptions
+boolBatteryBSOCAll: Boolean
+optBatteryModule1: FieldOptions
+boolBatteryModule1: Boolean
+optBatteryModule2: FieldOptions
+boolBatteryModule2: Boolean
+optBatteryCell: FieldOptions
+optBatteryCellSwitch: FieldOptions
+dblBatteryCellPosition: Double
+boolBatteryCellVoltage: Boolean
+intDimW: Integer
+intDimH: Integer
+LoadBatteryPanel(Pan...
+UpdateBatteryPanel(Pan...
+DrawBatteryPackCell(intIndex: Byte...
+DrawBatteryStatusPanel(optField: ...

```

```

<<Module>>
mdlPanel
+optPanel: PanelOptions
+strSettings: String
+optPanelMoveLeft: FieldOptions
+optPanelMoveRight: FieldOptions
+intDimW: Integer
+intDimH: Integer
+LoadPanel(Pan...
+SwitchPanel(intIndex: Byte, picOu...
+LoadSmallPanel(Pan...
+DrawSeparator(Left: Double, Top...
+DrawHeader(Left: Double, Top...
+DrawUnit(Left: Double, Top...
+CommandButton(Left: Double, To...
+PanelSwitchMouseOver(optField...
+DrawPanelMoveArrow(optField: F...
+PanelHeader(strSettings: String)...
+NextPanel(intIndex: Byte, Right: ...

```

```

<<Module>>
mdlPanelSatellite
+optSatelliteTime: CheckBoxOptions
+optSatelliteMode: CheckBoxOptions
+optSatellitePlay: FieldOptions
+optSatellitePrevious: FieldOptions
+optSatelliteNext: FieldOptions
+boolSatellitePlay: Boolean
+intSatelliteIndex: Integer
+intDimW: Integer
+intDimH: Integer
+datMTSATDate: Date
+strMTSATImage: String
+LoadSatellitePanel(Pan...
+UpdateSatellitePanel(Pan...
+LonToX(dblLongitude: Double): Dou...
+LatToY(dblLatitude: Double): Dou...
+LoadMTSAT()
+DrawPlayButton(optField: FieldO...
+DrawNextButton(optField: FieldO...
+DrawPreviousButton(optField: Fie...

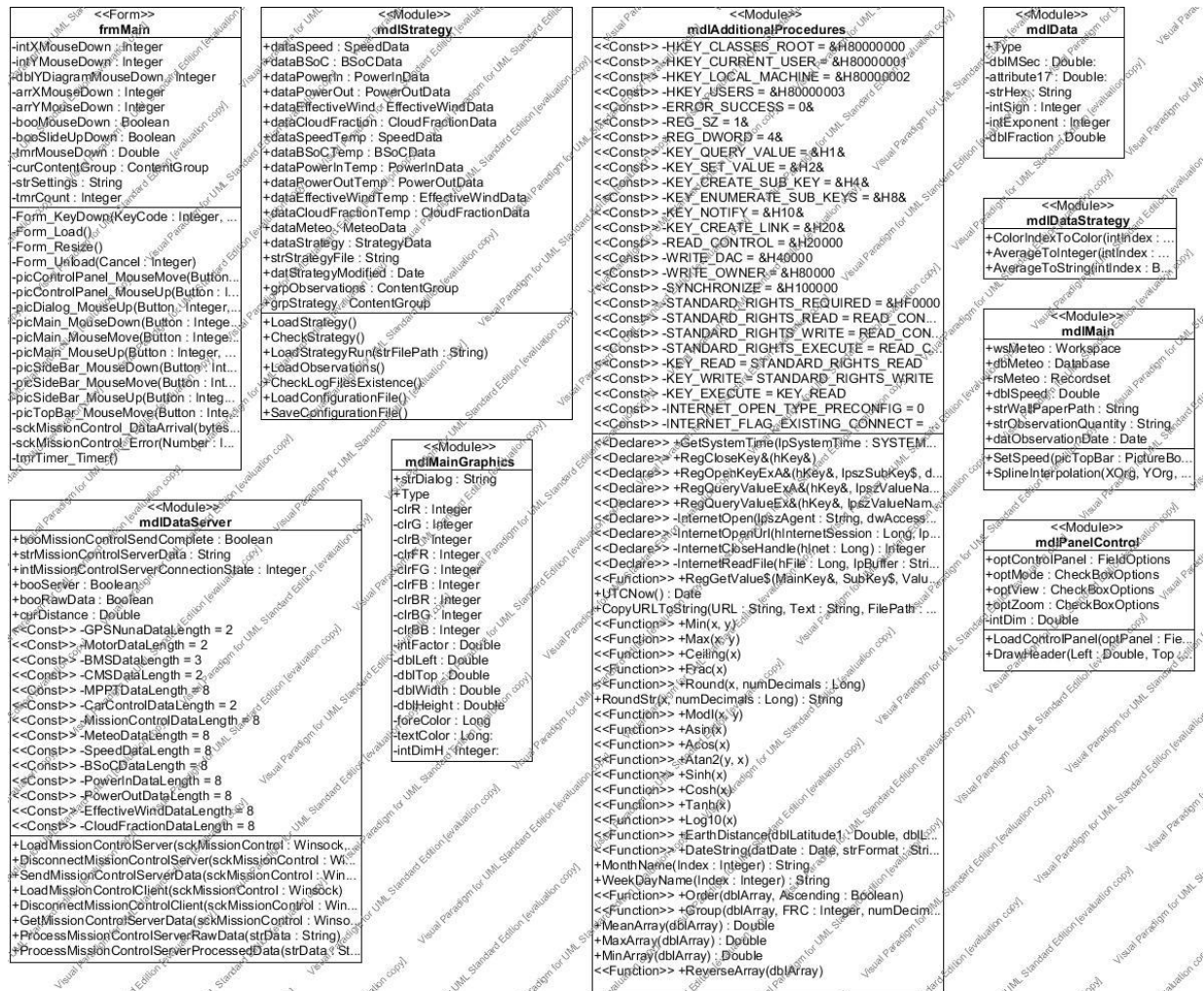
```

```

<<Module>>
mdlAdditionalGraphics
+Blend: Long
+bf: BLENDFUNCTION
+rc: TRANSPARENTRECT
+transWidth: Integer
+Const: +SRCCOPY = &HCC0020
+Const: +pi: Double = 3.14159265358979
+Const: +AC_SRC_OVER = &H0
+Const: +AC_SRC_ALPHA = &H1
+Const: +STRETCH_DELETESCANS = 3
+Const: +STRETCH_HALFTONE = 4
+Ret: Long
+W: Integer
+H: Integer
+MonoMaskDC: Long
+hMonoMask: Long
+hMonoInDC: Long
+hMonoInV: Long
+ResultDstDC: Long
+hResultDst: Long
+ResultSrcDC: Long
+hResultSrc: Long
+hPrevMask: Long
+hPrevInV: Long
+hPrevSrc: Long
+hPrevDst: Long
+OldBC: Long
+Radius: Single
+Theta: Single
+clr: Long
+B: Integer
+G: Integer
+R: Integer
+clrR: Integer
+clrG: Integer
+clrB: Integer
+clrRGB: Long
+srcx: Single
+srcy: Single
+bm: Bitmap
+ImageData: Byte
+OutputData: Byte
+x: Single
+y: Single
+clr1R: Integer
+clr1G: Integer
+clr1B: Integer
+clr2R: Integer
+clr2G: Integer
+clr2B: Integer
+clr3R: Integer
+clr3G: Integer
+clr3B: Integer
+clr4R: Integer
+clr4G: Integer
+clr4B: Integer
+S: Single
+t: Single
+u: Single
+v: Single
+k: Single
+L: Single
+m: Single
+n: Single
+clrBG: Long
+clrFG: Long
+clRBG: Integer
+clRBG: Integer
+clRBGR: Integer
+clRGB: Integer
+clRGB: Integer
+clFG: Integer
+clFGR: Integer
+clFactor: Double
+attribute: Integer
+attribute2: Integer
+attribute3: Integer
+Declare: +GetDC(hWnd: Long): Long
+Declare: +BitBlt(hdcDest: Long, x: Lo...
+Declare: +ReleaseDC(hWnd: Long, hD...
+Declare: +SetPixel(hdc: Long, x: Long...
+Declare: +GetPixel(hdc: Long, x: Lon...
+Declare: +CopyMemory(lpDest: Any, l...
+Declare: +AlphaBlend(hdcDest: Long, ...
+Declare: +CreateBitmap(nWidth: Long, ...
+Declare: +SetBkColor(hdc: Long, crCo...
+Declare: +SelectObject(hdc: Long, hO...
+Declare: +CreateCompatibleBitmap(hdc...
+Declare: +CreateCompatibleDC(hdc: ...
+Declare: +DeleteDC(hdc: Long): Long...
+Declare: +DeleteObject(hObject: Long)...
+Declare: +StretchBlt(hdc: Long, nC...
+Declare: +SetStretchBltMode(hdc: Lon...
+Declare: +GetObject(hObject: Long, nC...
+Declare: +GetBitmapBits(hBitmap: Lon...
+Declare: +SetBitmapBits(hBitmap: Lon...
+Declare: +PlgBlt(hdcDest: Long, lpPoin...

```

APPENDIX IV: CURRENT APPLICATION - CLASS DIAGRAM - STRATEGY



```

<<Module>>
mdlAdditionalGraphics
+Blend : Long
+bf : BLENDFUNCTION
+rc : TRANSPARENTRECT
+transWidth : Integer
<<Const>> +SRC_COPY = &H000020
<<Const>> +pl : Double = 3.14159265358979
<<Const>> +AC_SRC_OVER = &H0
<<Const>> +AC_SRC_ALPHA = &H1
<<Const>> +STRETCH_DELETESCANS = 3
<<Const>> +STRETCH_HALFTONE = 4
-nRet : Long
-W : Integer
-H : Integer
-MonoMaskDC : Long
-hMonoMask : Long
-hMonoInVDC : Long
-hMonoInV : Long
-ResultDstDC : Long
-hResultDst : Long
-ResultSrcDC : Long
-hResultSrc : Long
-hPrevMask : Long
-hPrevInv : Long
-hPrevSrc : Long
-hPrevDst : Long
-OldBC : Long
-Radius : Single
-Theta : Single
-clr : Long
-B : Integer
-G : Integer
-R : Integer
-clrR : Integer
-clrG : Integer
-clrB : Integer
-clrRGB : Long
-srgr : Single
-srgr : Single
-bm : Bitmap
-ImageData : Byte
-OutputData : Byte
-x : Single
-y : Single
-clr1R : Integer
-clr1G : Integer
-clr1B : Integer
-clr2R : Integer
-clr2G : Integer
-clr2B : Integer
-clr3R : Integer
-clr3G : Integer
-clr3B : Integer
-clr4R : Integer
-clr4G : Integer
-clr4B : Integer
-S : Single
-t : Single
-u : Single
-v : Single
-k : Single
-L : Single
-m : Single
-n : Single
-clrBG : Long
-clrFG : Long
-clrBGB : Integer
-clrBGG : Integer
-clrBGR : Integer
-clrFGB : Integer
-clrFGG : Integer
-clrFGR : Integer
-clrFactor : Double
-attribute : Integer
-attribute2 : Integer
-attribute3 : Integer

<<Declare>> +GetDC(hWnd : Long) : Long
<<Declare>> +BitBlt(hDestDC : Long, x : Long, y : Long, w : Long, h : Long, hSrcDC : Long, xSrc : Long, ySrc : Long, dwRop : Long) : Long
<<Declare>> +ReleaseDC(hWnd : Long, hDC : Long) : Long
<<Declare>> +SetPixel(hDC : Long, x : Long, y : Long, clr : Long) : Long
<<Declare>> +GetPixel(hDC : Long, x : Long, y : Long) : Long
<<Declare>> +CopyMemory(lpvDest : Any, lpvSrc : Any, cb : Long) : Long
<<Declare>> +AlphaBlend(hdcDest : Long, nXDest : Long, nYDest : Long, nWidth : Long, nHeight : Long, hdcSrc : Long, nXSrc : Long, nYSrc : Long, nWidth : Long, nHeight : Long, dwBlendOp : Long) : Long
<<Declare>> +CreateBitmap(nWidth : Long, nHeight : Long, nPlanes : Long, nBitsPerPixel : Long, lpvBits : Long) : Long
<<Declare>> +SetBkColor(hDC : Long, clrBk : Long) : Long
<<Declare>> +SelectObject(hDC : Long, hObject : Long) : Long
<<Declare>> +CreateCompatibleBitmap(hDC : Long, nWidth : Long, nHeight : Long) : Long
<<Declare>> +CreateCompatibleDC(hDC : Long) : Long
<<Declare>> +DeleteDC(hDC : Long) : Long
<<Declare>> +DeleteObject(hObject : Long) : Long
<<Declare>> +StretchBlt(hDC : Long, xDest : Long, yDest : Long, wDest : Long, hDest : Long, hSrcDC : Long, xSrc : Long, ySrc : Long, wSrc : Long, hSrc : Long, dwRop : Long) : Long
<<Declare>> +SetStretchBltMode(hDC : Long, nMode : Long) : Long
<<Declare>> +GetObject(hObject : Long, nCount : Long) : Long
<<Declare>> +GetBitmapBits(hBitmap : Long, dwCount : Long, lpvBits : Long) : Long
<<Declare>> +SetBitmapBits(hBitmap : Long, dwCount : Long, lpvBits : Long) : Long
<<Declare>> +PlgBlt(hdcDest : Long, lpPoint : Long, hBitmap : Long, dwCount : Long, lpvBits : Long) : Long

```

```

<<Module>>
mdlPanelStrategy
+optDiagram : DiagramOptions
+optPanel : PanelOptions
+statePanel : Byte
+booSlidePanel : Boolean
+booAutoZoom : Boolean
+booDistance : Boolean
+booXAxisMouseDown : Boolean
+booXZoomAxisMouseDown : Boolean
+booYAxisMouseDown : Boolean
+booYZoomAxisMouseDown : Boolean
+optDistancePanel : PanelOptions
+optSlideLeft : FieldOptions
+optSlideRight : FieldOptions
+onSlideLeft : Boolean
+onSlideRight : Boolean
-intDim : Double
-intAxisLeft : Double
-intAxisTop : Double
-intAxisWidth : Double
-intAxisHeight : Double
+optPanelTemp : PanelOptions
-dblT : Double
-dblX : Double
-dblY : Double
-curLeft : Double
-dblGridStep : Double
-XMin : Double
-XMax : Double
-XZoomMin : Double
-XZoomMax : Double
-XStep : Double
-XZoomStep : Double
-dblXNew : Double
-dblYNew : Double
-dblXPix : Double
-dblYPix : Double
-clr : Long
-numCount : Integer
-curRight : Double
-curTop : Double
-curBottom : Double
-curX : Double
-maxX : Double
-dblWeight : Double
-dblInterval : Double
-Function
-Sub
-dblMax : Double
-dblMin : Double
-dblBound : Double
-dblUBound : Double

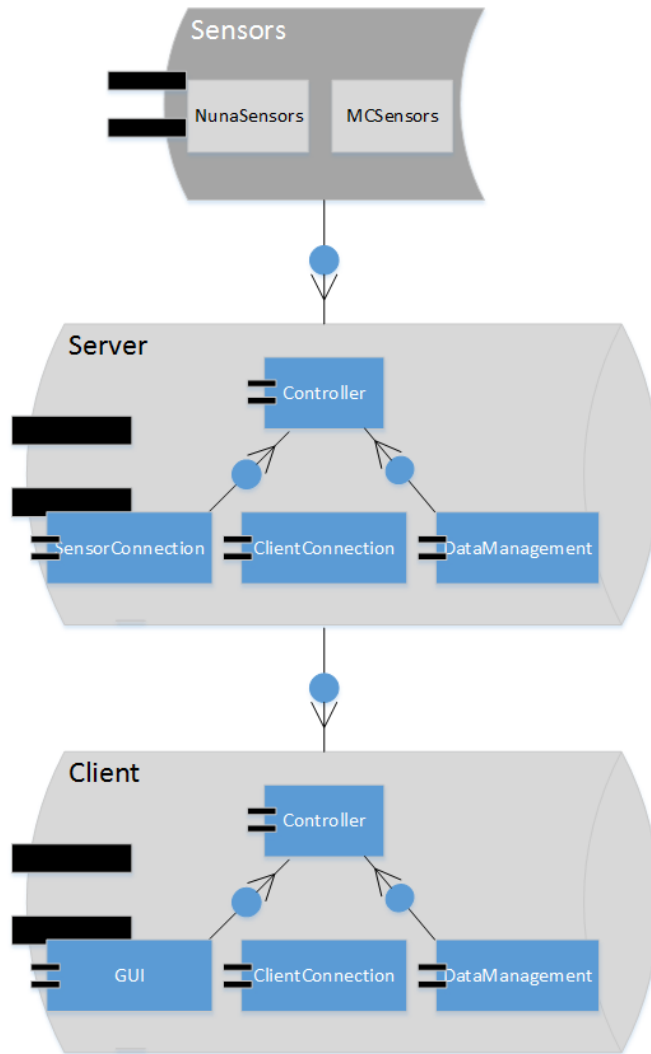
```

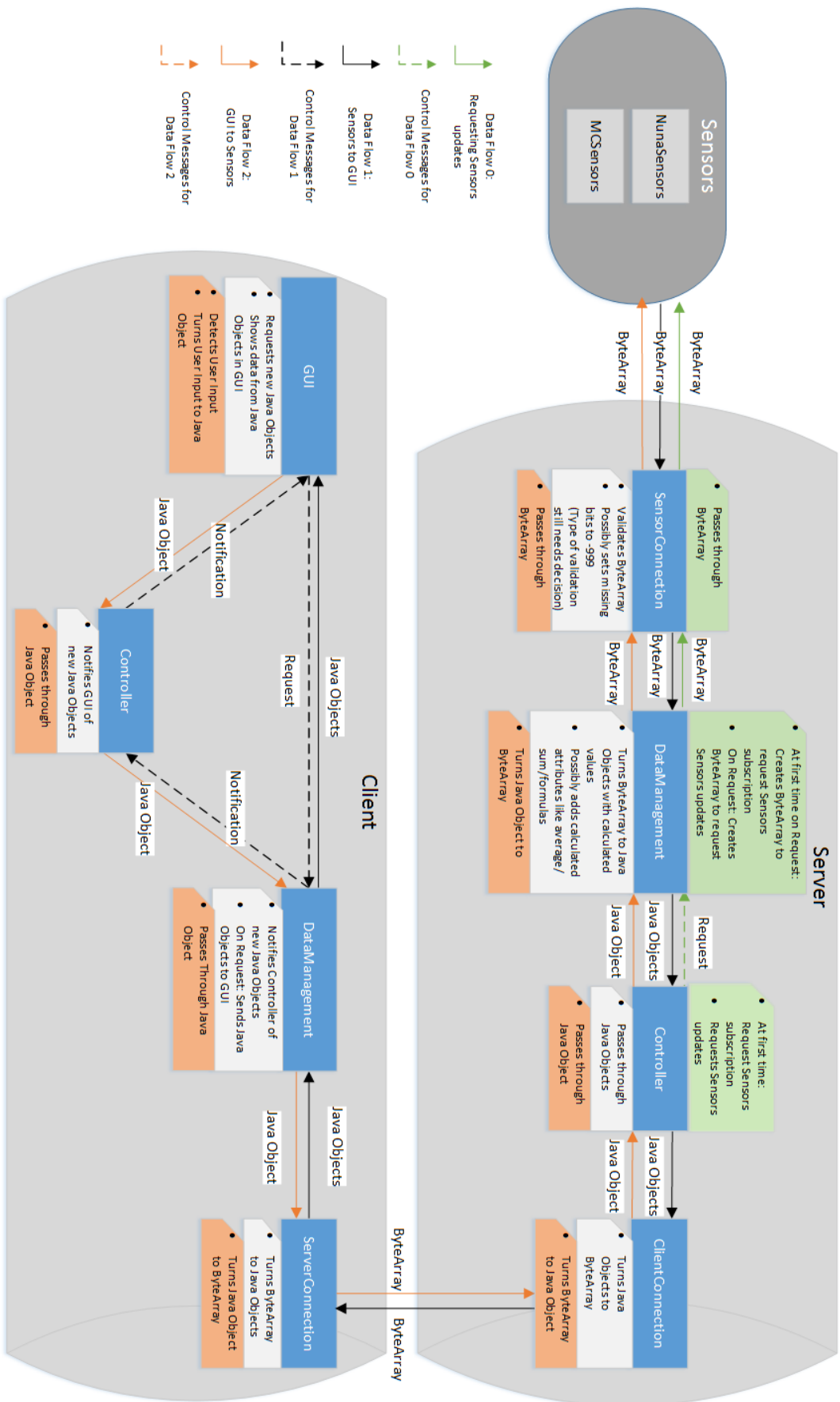
```

<<Module>>
mdlSideBar
+booSideBarScrollBar : Boolean
+dblSideBarContentTop : Double
+dblSideBarScrollBarSize : Double
+intSideBarLeft : Integer
+intSideBarTop : Integer
+intScrollBarWidth : Integer
+intSideBarWidth : Integer
+intSideBarHeight : Integer
+booScrollBarMouseDown : Boolean
+dblSideBarContentTopMouseDown : Double
+booFound : Boolean
+booExpandCollapseState : Integer
+booExpandCollapseStateOld : Integer
+boolItemEnabledState : Integer
+boolItemEnabledStateOld : Integer
+boolItemLeftRightState : Integer
+boolItemLeftRightStateOld : Integer
-intHeight : Double
-intWidth : Double
-clrBack : Long
-clrBack2 : Long
-dblRatio : Double

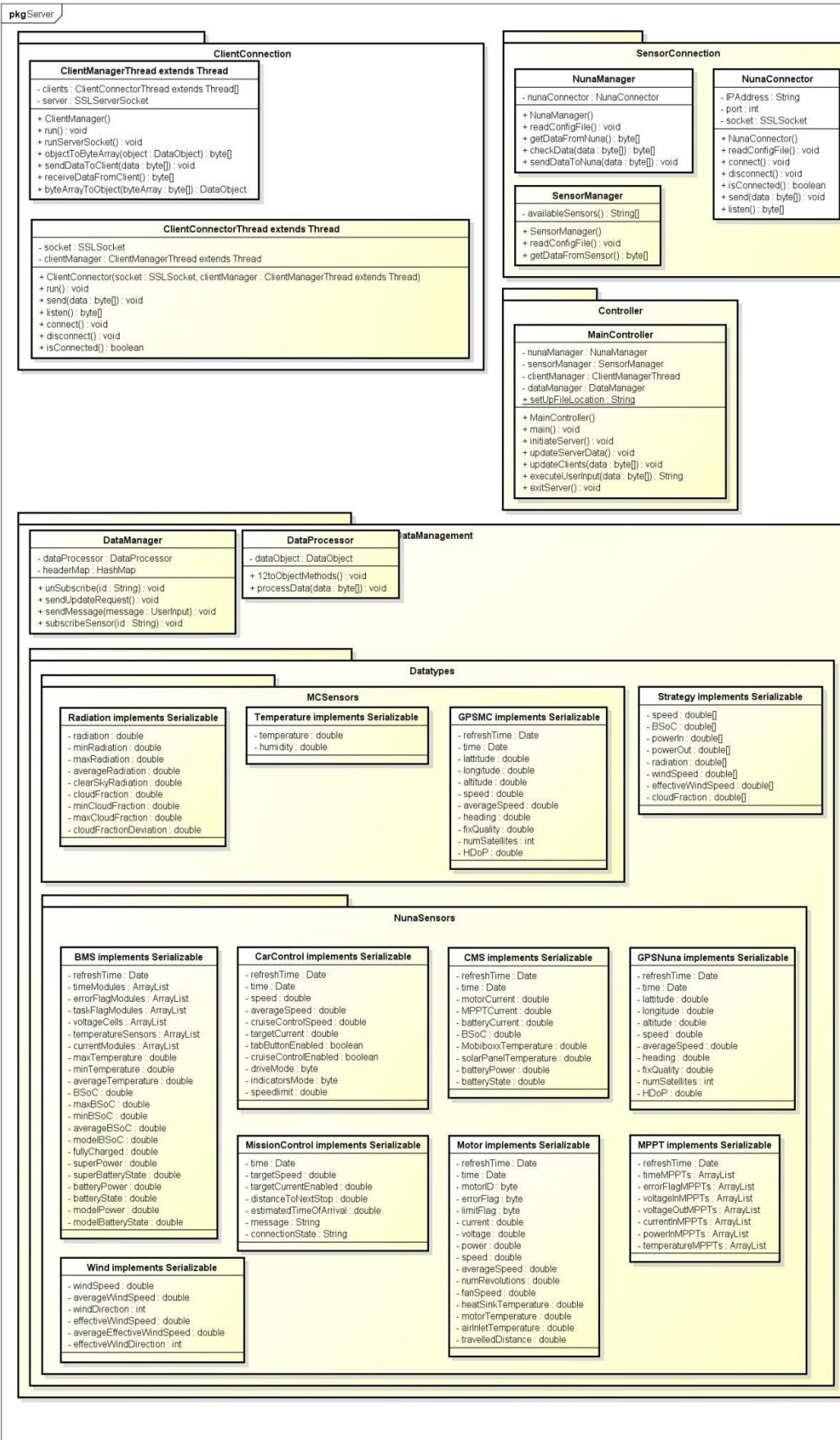
+LoadSideBar(picSideBar : PictureBox)
+LoadSideBarHeader(picSideBar : PictureBox, picMain : PictureBox)
+LoadSideBarContent(picSideBarContent : PictureBox, picMain : PictureBox)
+ShowSideBarContent(picSideBar : PictureBox, picMain : PictureBox)
+ShowSideBar(picSideBar : PictureBox, picMain : PictureBox)
+HideSideBar(picSideBar : PictureBox, picMain : PictureBox)
+DrawSideBarGroupHeader(picSideBarContent : PictureBox, picMain : PictureBox)
+DrawSideBarGroupFooter(picSideBarContent : PictureBox, picMain : PictureBox)
+DrawSideBarItemObservations(picSideBarContent : PictureBox, picMain : PictureBox)
+ExpandCollapse(Left : Double, Top : Double, Width : Double, Height : Double) : Integer
+LeftRight(Left : Double, Top : Double, Width : Double, Height : Double) : Integer
+CheckExpandCollapse(X : Single, Y : Single, Action : Integer) : Integer
+SetExpandCollapse(intIndex : Integer, Action : Integer) : Integer
+ContentExpandCollapse(intIndex : Integer, intIndex : Integer, intIndex : Integer) : Integer
+CheckItemEnabled(X : Single, Y : Single, Action : Integer) : Integer
+SetItemEnabled(intIndex : Integer, Action : Integer) : Integer
+CheckItemLeftRight(X : Single, Y : Single, Action : Integer) : Integer
+SetItemLeftRight(intIndex : Integer, Action : Integer) : Integer
+ContentItemLeftRight(intIndex : Integer, grpContent : Integer) : Integer
+DrawPlotStyle(Left : Double, Top : Double, Width : Double, Height : Double) : Integer

```

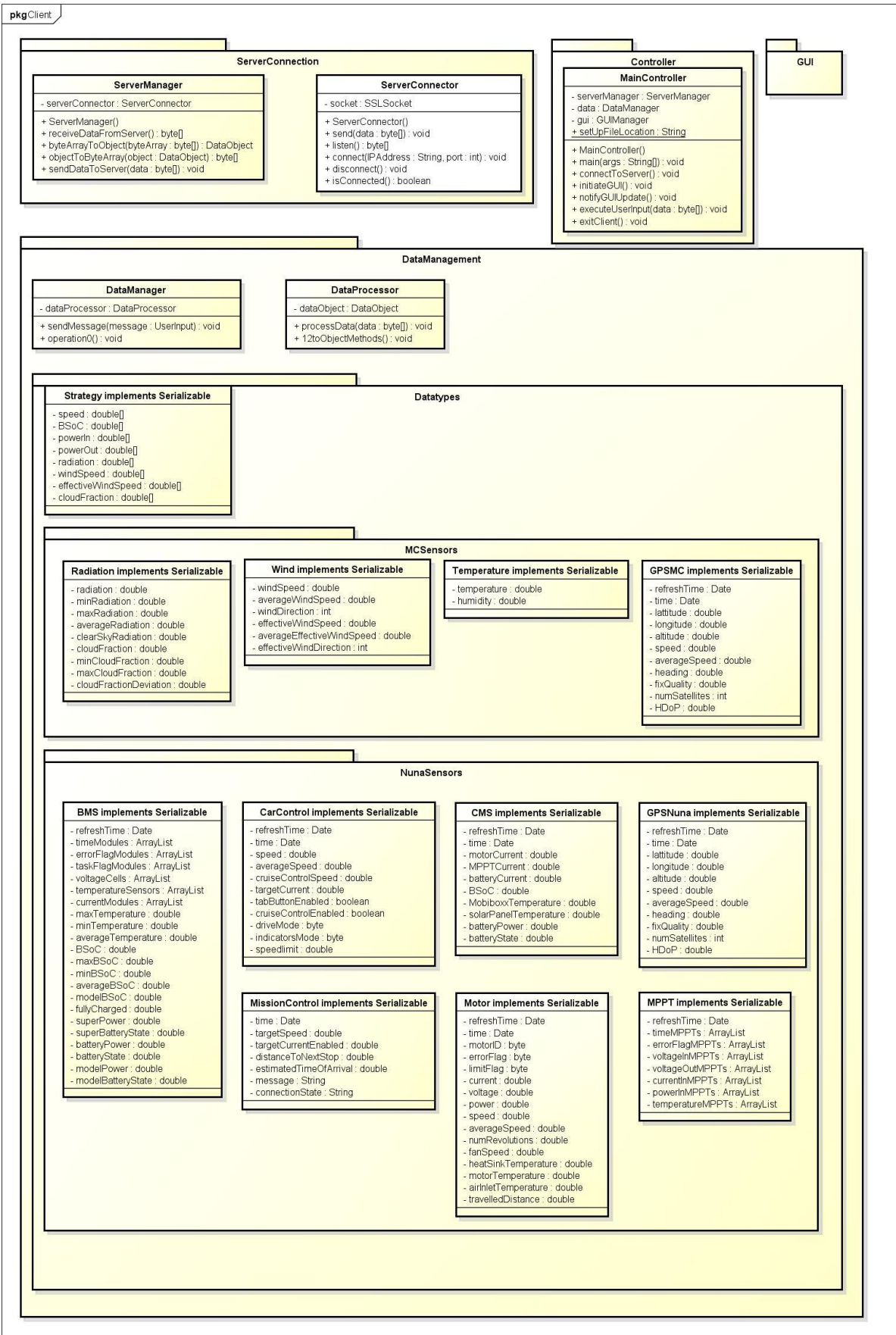




APPENDIX VII: NEW APPLICATION - CLASS DIAGRAM - SERVER

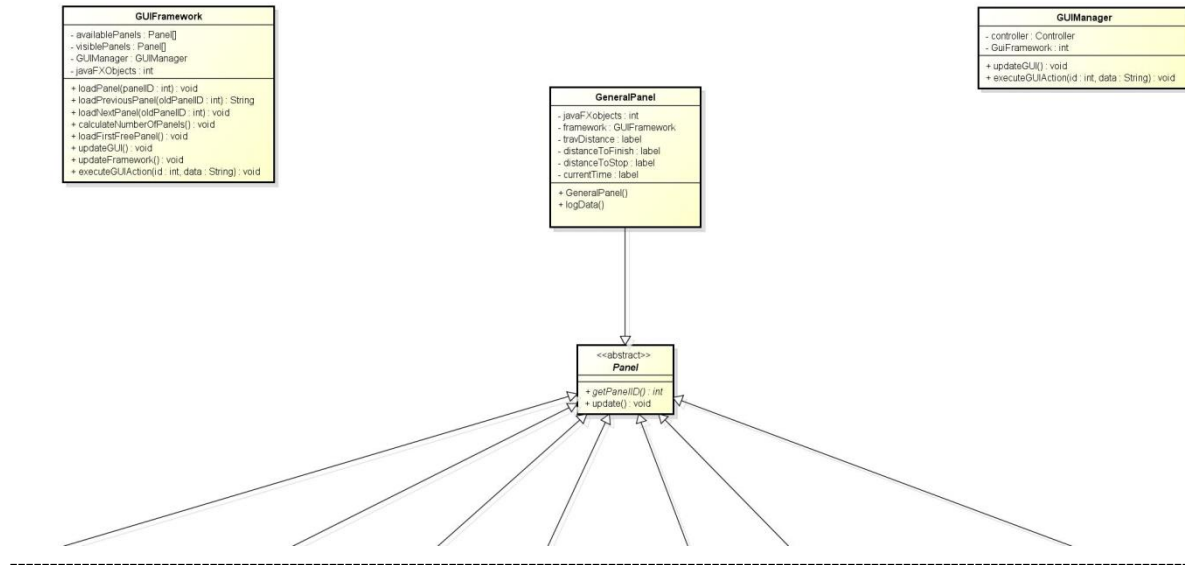


APPENDIX VIII: NEW APPLICATION - CLASS DIAGRAM - CLIENT

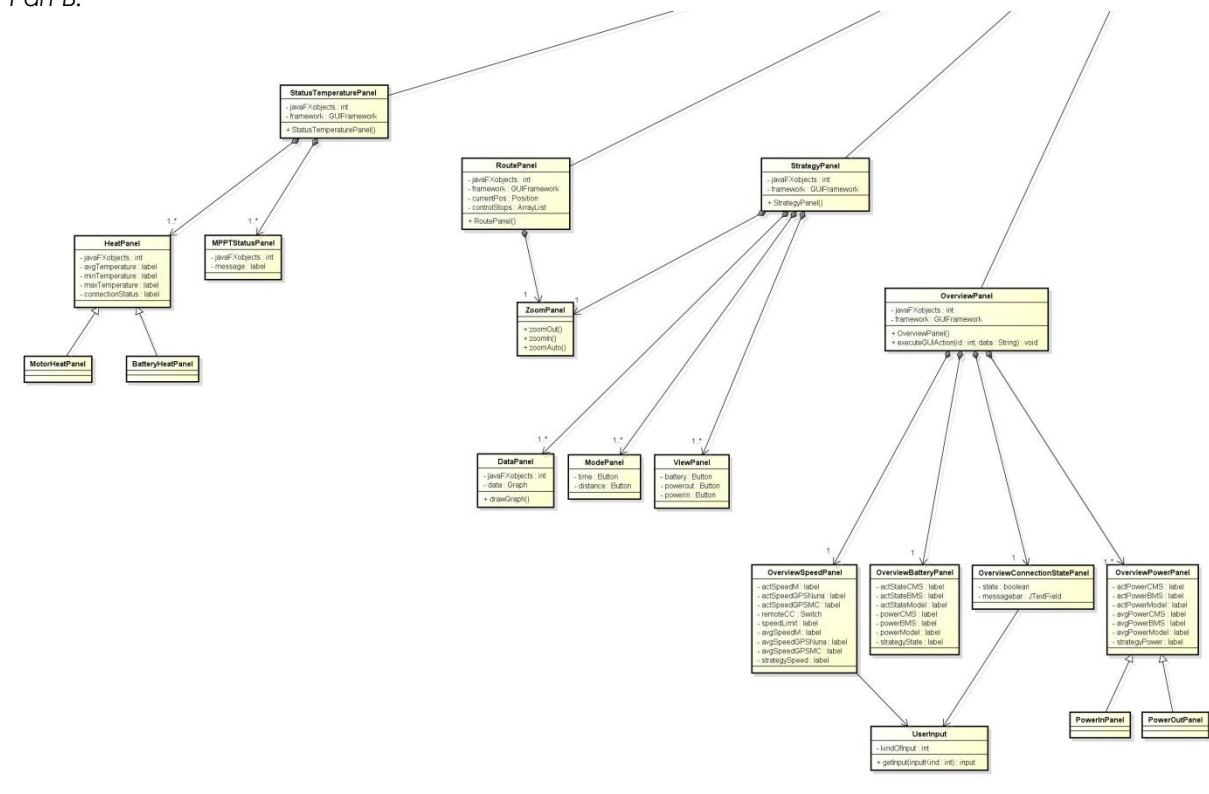


APPENDIX VIII: NEW APPLICATION - CLASS DIAGRAM - GUI

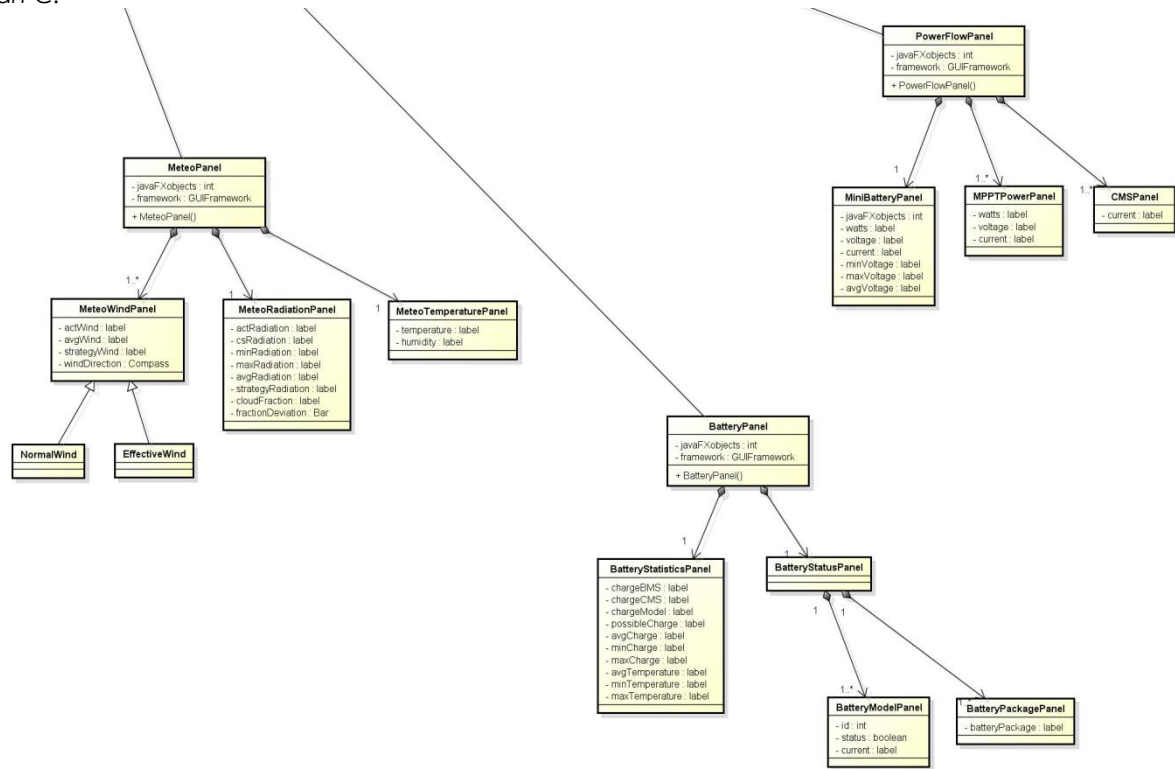
Part A:



Part B:



Part C:



REFERENCES

- [1] Front page picture of Nuna 7, as found 20-03-2014 on
<http://ietjedesign.com/wp-content/uploads/2013/03/NUNA7-schets-infographic-2.jpg>
- [2] As found 03-04-2014 on
http://upload.wikimedia.org/wikipedia/commons/f/ff/LabVIEW_Block_diagram.JPG

APPENDIX 3: PROJECT LOGS

JOURNAL OF TASKS

	Research phase			Exams	Development phase										Test phase		Extension
Product Backlog / Sprint week	11	12	13	14-16	17	18	19	20	21	22	23	24	25	26	27	28	
Documenting List of Requirements																	
Documenting Plan of Action Report																	
Reverse engineering of current application																	
Researching Java GUI frameworks																	
Researching Java real-time connections																	
Documenting Research Phase report																	
Modelling the new application																	
Creating a framework																	
Connecting application to mobiboxx																	
Sending, requesting mobiboxx data																	
Showing data in some way in framework																	
Creating overview panel																	
Creating battery panel																	
Creating power flow panel																	
Testing application with driving solar car																	
Creating strategy panel																	
Creating remaining panels																	
Sending code to SIG for evaluation																	
Creating remaining features																	
Implementing SIG code feedback																	
Composing Final Report																	
Testing thoroughly																	
Booking presentation room and inform																	
Submit Final Report to BEP coordinators + Nicolas + cliënt																	
Resending code to SIG																	
Presenting the final product																	
Extra functionalities																	

WORK HOURS

			EEF	JUR	ZAK	ISH	LO
Total			404,00	432,50	393,50	446,00	399,00
Week 1	Monday	21-apr-14			0,00		(2de paasdag)
	Tuesday	22-apr-14	8,00	8,00	6,50	8,00	8,00
	Wednesday	23-apr-14	7,00	9,00	7,00	7,00	7,00
	Thursday	24-apr-14	9,50	8,00	7,00	8,00	8,00
	Friday	25-apr-14	7,00	9,00	9,00	8,00	4,50
	Saturday	26-apr-14					
	Sunday	27-apr-14			1,00	1,00	
Week 2	Monday	28-apr-14	8,00	7,00	5,50	10,00	0,00
	Tuesday	29-apr-14	8,00	8,00	7,50	8,00	8,00
	Wednesday	30-apr-14	8,00	8,00	8,50	9,00	9,00
	Thursday	1-mei-14	9,00	9,00	8,00	9,00	9,00
	Friday	2-mei-14	8,00	8,00	8,50	8,00	8,00
	Saturday	3-mei-14				3,00	
	Sunday	4-mei-14					
Week 3	Monday	5-mei-14	1,50	4,00	8,00		4,00 (Bevrijdingsdag)
	Tuesday	6-mei-14	8,50	8,50	8,50	8,50	8,50
	Wednesday	7-mei-14	6,00	9,00	9,00	9,00	11,00
	Thursday	8-mei-14	9,00	9,00	9,00	9,00	8,00

	Friday	9-mei-14	8,50	10,00	8,50	6,50	8,50
	Saturday	10-mei-14	3,00	3,00	3,00	7,50	
	Sunday	11-mei-14			7,50	3,00	
<u>Week 4</u>	Monday	12-mei-14	9,00	9,00	9,00	8,00	9,00
	Tuesday	13-mei-14	8,00	8,50	0,00	8,00	8,50
	Wednesday	14-mei-14	8,00	8,00	8,00	8,00	8,00
	Thursday	15-mei-14	8,50	8,50	8,50	8,50	8,50
	Friday	16-mei-14	9,00	9,00	9,00	9,00	8,00
	Saturday	17-mei-14		4,50			
	Sunday	18-mei-14	2,00	2,00	4,00	4,00	8,00
<u>Week 5</u>	Monday	19-mei-14	8,50	8,50	8,50	7,50	8,50
	Tuesday	20-mei-14	8,50	8,50	4,50	8,50	8,50
	Wednesday	21-mei-14	8,50	10,00	8,50	8,50	8,50
	Thursday	22-mei-14	8,00	8,00	8,00	8,00	8,00
	Friday	23-mei-14	7,50	7,50	7,50	4,50	7,50
	Saturday	24-mei-14	1,00	1,00	3,50	2,00	
	Sunday	25-mei-14	2,00		1,50	1,00	
<u>Week 6</u>	Monday	26-mei-14	10,00	10,00	10,00	9,50	10,00
	Tuesday	27-mei-14	7,50	7,50	7,50	7,50	7,50
	Wednesday	28-mei-14	8,00	8,50	8,50	8,00	10,50
	Thursday	29-mei-14			2,00		<i>(Hemelvaartsdag)</i>
	Friday	30-mei-14		2,00	7,00		<i>(Collectieve vrije dag)</i>
	Saturday	31-mei-14					2,00
	Sunday	1-jun-14			2,00		
<u>Week 7</u>	Monday	2-jun-14	8,50	8,50	8,50	8,00	8,50
	Tuesday	3-jun-14	8,00	9,00	8,00	11,00	8,00
	Wednesday	4-jun-14	7,50	7,50	7,50	7,50	7,50
	Thursday	5-jun-14	7,50	10,00	8,50	10,00	10,00
	Friday	6-jun-14	8,50	8,00	8,00	5,50	8,50
	Saturday	7-jun-14		3,00			
	Sunday	8-jun-14	2,00			5,50	
<u>Week 8</u>	Monday	9-jun-14		1,50	6,50	2,00	<i>(Pinksteren)</i>
	Tuesday	10-jun-14	9,00	8,00	8,50	10,00	8,50
	Wednesday	11-jun-14	11,50	12,00	9,00	12,00	11,00
	Thursday	12-jun-14	10,00	10,00	8,00	12,50	9,00
	Friday	13-jun-14	9,00	9,00	9,00	6,50	8,00
	Saturday	14-jun-14		6,50	5,00		
	Sunday	15-jun-14	4,00		3,50	4,00	6,00
<u>Week 9</u>	Monday	16-jun-14	10,00	10,50	0,00	12,00	8,00

	Tuesday	17-jun-14	10,00	10,00	8,00	12,00	8,00
	Wednesday	18-jun-14	12,50	11,50	8,50	12,50	8,00
	Thursday	19-jun-14	9,00	10,50	0,00	9,00	11,00
	Friday	20-jun-14	9,00	8,50	0,00	8,50	9,00
	Saturday	21-jun-14	2,00	5,00	4,50	4,00	
	Sunday	22-jun-14	7,00	1,00	4,50	9,00	8,00
<u>Week 10</u>	Monday	23-jun-14	8,00	10,00	8,00	10,00	8,00
	Tuesday	24-jun-14	11,00	12,50	10,50	14,50	11,50
	Wednesday	25-jun-14	8,50	13,00	11,50	14,50	11,50
	Thursday	26-jun-14	8,50	8,50	8,50	12,50	8,50
	Friday	27-jun-14					
	Saturday	28-jun-14					
	Sunday	29-jun-14					

APPENDIX 4: TESTING PLAN

TEST ALL VALUES DISPLAYED IN GUI

OVERVIEW

- ☐ Actual speed
- ☐ Actual motor speed
- ☐ Actual GPS speed
- ☐ Actual MC speed
- ☐ Average speed
- ☐ Average motor speed
- ☐ Average GPS speed
- ☐ Average MC speed
- ☐ Speed limit
- ☐ Strategy speed
- ☐ BSoC
- ☐ CMS BSoC
- ☐ BMS BSoC
- ☐ Model BSoC
- ☐ Battery power
- ☐ CMS battery power
- ☐ BMS battery power
- ☐ Model battery power
- ☐ Strategy battery
- ☐ Actual power in
- ☐ Actual CMS power in
- ☐ Actual MPPT power in
- ☐ Actual model power in
- ☐ Average power in
- ☐ Average CMS power in
- ☐ Average MPPT power in
- ☐ Average model power in
- ☐ Strategy power in
- ☐ Actual power out
- ☐ Actual CMS power out
- ☐ Actual MPPT power out
- ☐ Actual model power out
- ☐ Average power out
- ☐ Average CMS power out
- ☐ Average MPPT power out
- ☐ Average model power out
- ☐ Strategy power out

BATTERY

- ☐ BSoC
- ☐ CMS BSoC
- ☐ Model BSoC
- ☐ Total voltage
- ☐ Average voltage
- ☐ Min voltage
- ☐ Max voltage
- ☐ Min temperature
- ☐ Average temperature
- ☐ Max temperature
- ☐ Voltages and colours cells 1-27
- ☐ Temperatures and colours cells 1-27

POWERFLOW

- ☐ MPPT 1-5 power and colour
- ☐ MPPT 1-5 current and colour

- ☐ MPPT 1-5 voltage and colour
- ☐ MPPT 6 (concentrator) power and colour
- ☐ MPPT 6 (concentrator) current and colour
- ☐ MPPT 6 (concentrator) voltage and colour
- ☐ Battery percentage charged and colour
- ☐ Battery power
- ☐ Battery current
- ☐ Battery voltage
- ☐ Battery min voltage
- ☐ Battery average voltage
- ☐ Battery max voltage
- ☐ CMS battery current
- ☐ CMS MPPT current
- ☐ CMS motor current
- ☐ Motor power and colour
- ☐ Motor current and colour
- ☐ Motor voltage and colour

STATUS

- ☐ MPPT 1-5 status and colour
- ☐ MPPT 1-5 temperature and colour
- ☐ MPPT 6 (concentrator) status and colour
- ☐ MPPT 6 (concentrator) temperature and colour
- ☐ Battery min temperature
- ☐ Battery average temperature
- ☐ Battery Max temperature
- ☐ Battery status module 1 and colour
- ☐ Battery status module 2 and colour
- ☐ BMS colour
- ☐ Car control colour
- ☐ CMS colour
- ☐ GPS (Nuna) colour
- ☐ Motor colour
- ☐ MPPT colour
- ☐ GPS (Mission Control) colour
- ☐ Pyrano colour
- ☐ Thermo colour
- ☐ Wind colour
- ☐ Motor temperature and colour
- ☐ Motor air inlet temperature and colour
- ☐ Motor heat sink temperature and colour
- ☐ Motor status and colour

METEO

- ☐ Actual radiation
- ☐ Average radiation
- ☐ Strategy radiation
- ☐ Cloud fraction
- ☐ Clear sky radiation
- ☐ Temperature
- ☐ Humidity
- ☐ Actual effective wind speed and colour
- ☐ Actual effective wind direction
- ☐ Average effective wind speed and colour
- ☐ Average effective wind direction
- ☐ Strategy effective wind speed and colour
- ☐ Strategy effective wind direction
- ☐ Actual normal wind speed and colour
- ☐ Actual normal wind direction
- ☐ Average normal wind speed and colour
- ☐ Average normal wind direction
- ☐ Strategy normal wind speed and colour

- ☐ Strategy normal wind direction

ROUTE

- ☐ Current location
- ☐ Route

HISTORY

No values to test

TEST IF ALL DATA IS LOGGED CORRECTLY

SENSOR LOGS

- ☐ BMS
- ☐ CarControl
- ☐ CMS
- ☐ GPSNuna
- ☐ Motor
- ☐ MPPT
- ☐ GPSMC
- ☐ Radiation
- ☐ Temperature
- ☐ Wind

DATAREPOSITORY LOGS

- ☐ BMS
- ☐ CarControl
- ☐ CMS
- ☐ GPSNuna
- ☐ MissionControl
- ☐ Model
- ☐ Motor
- ☐ MPPT
- ☐ GPSMC
- ☐ Radiation
- ☐ Temperature
- ☐ Wind

TEST STABILITY OF THE CONNECTION BETWEEN THE SERVER AND THE MOBIBOXX

- ☐ Start server via GUI and check if data is received.
- ☐ Start server via GUI. Move out of reach of Mobiboxx so that no data is received anymore. Move back in range of Mobiboxx and check if data is received.
- ☐ Start server via GUI. Disconnect from HankDeTank. Connect to HankDeTank and check if data is received.
- ☐ Start two servers via different GUIs and check if data is properly received (by both, or, if only one server receives data, test if the other server receives data when the receiving one is closed).

TEST STABILITY OF THE CONNECTION BETWEEN THE SERVER AND THE CLIENTS

- ☐ Start two GUIs on different machines. Start the server via one, check if both GUIs receive server data.
- ☐ Start five GUIs on different machines. Start server via one, check if all servers receive data **without delay**.
- ☐ Start two GUIs on different machines. Start server via one. Move one machine out of reach of the network. Move back into range and check if data received.
- ☐ Start one GUI. Open a second window from the first GUI. Check if both windows receive the same data at the same time.

TEST TONUNA FUNCTIONALITY

- ☐ Start GUI and a server. Send a chat message to the Nuna and check if message is received.
- ☐ Test if remote cruise control is disabled when cruise control is turned off.

- ☐ Send a remote cruise control speed to Nuna when cruise control is enabled, and check if Nuna drives speed of remote cruise control
- ☐ Start two GUIs on different machines. Send remote cruise control speed to Nuna and check if all GUIs display the sent remote cruise control speed.

TEST DAYS

TEST DAY 1: TESTING WITH MC SENSORS (25-26/06)

- ☐ All values of MC sensors displayed in GUI
- ☐ All data of MC sensors logged correctly
- ☐ Stable connection between server and sensors

TEST DAY 2: TESTING WITH STATIONARY NUNA (25-26/06)

- ☐ All values of Nuna sensors displayed in GUI
- ☐ All data of Nuna sensors logged correctly
- ☐ Stable connection between server and Mobiboxx
- ☐ Stable connection between server and clients
- ☐ Working toNuna functionality

TEST DAY 3: DRIVING NUNA IN PARKING PLACE (25-26/06)

- ☐ All values displayed in GUI
- ☐ All data logged correctly
- ☐ Stable connection between server and Mobiboxx
- ☐ Stable connection between server and clients
- ☐ Working toNuna functionality

TEST DAY 4: OFFICIAL TEST DAY WITH NUNA 7 (29/06)

- ☐ All values displayed in GUI
- ☐ All data logged correctly
- ☐ Stable connection between server and Mobiboxx
- ☐ Stable connection between server and clients
- ☐ Working toNuna functionality

More to follow.